

Project Documentation

for

MiniContaBill

Version 1.0 approved

Prepared by

Bran Ioana-Andreea,

Chiuariu Silviu-Vasile,

Negoită Petru,

Zabara Sonia

**Universitatea Tehnică Gheorghe Asachi Iași, Facultatea de Automatică și
Calculatoare, Calculatoare și Tehnologia Informației, grupa 1310A**

2025

1. Introduction.....	4
2. Software Requirements Specifications.....	4
1. Introduction.....	4
1.1 Purpose.....	4
1.2 Document Conventions.....	4
1.3 Intended Audience and Reading Suggestions.....	4
1.4 Product Scope.....	5
1.5 References.....	5
2. Overall Description.....	5
2.1 Product Perspective.....	5
2.2 Product Functions.....	6
2.3 User Classes and Characteristics.....	7
2.4 Operating Environment.....	7
2.5 Design and Implementation Constraints.....	8
2.6 User Documentation.....	8
2.7 Assumptions and Dependencies.....	8
3. External Interface Requirements.....	9
3.1 User Interfaces.....	9
3.2 Hardware Interfaces.....	9
3.3 Software Interfaces.....	9
3.4 Communications Interfaces.....	9
4. System Features.....	10
4.1 Capital Module.....	10
4.2 Production Cost and Profit Module.....	11
4.3 Demand and Supply Module.....	13
4.4 Salaries and Employees Module.....	15
4.5 Performance Requirements.....	17
4.6 Safety Requirements.....	18
4.7 Security Requirements.....	19
4.8 Software Quality Attributes.....	19
4.9 Business Rules.....	20
5. Other Requirements.....	20
Appendix A: Glossary.....	20
3. Analysis Models.....	22
3.1 Class Diagrams.....	22
3.2 UseCase Diagrams.....	25
3.3 Activity Diagrams.....	26
3.4 Sequences Diagrams.....	30

4. Design Patterns.....	34
4.1 Builder (Company Object Construction).....	34
4.2 Strategy.....	35
5. Testing.....	37
5.1 Purpose of testing.....	37
5.2 Methodology.....	37
5.3 Test Cases.....	38
5.4 Conclusion.....	42
7. User Guide.....	42
7.1 Help file organization (CHM).....	42
7.2 Overview.....	42
7.3 Step-by-step Guides.....	43
7.3.1 Company selection.....	45
7.3.2 Company Attributes.....	46
7.3.3 Dashboard.....	46
7.4 Troubleshooting.....	52
8. Conclusions.....	54
9. Possible future developments.....	54
Appendix 1 - Relevant code snippets.....	55
A1. Depreciation calculation (3 methods: linear, progressive, declining balance).....	55
A2. Dividend calculation (only for SA).....	56
A3. Calculating price elasticity of demand.....	57
A4. Calculating Net Salary.....	58
A5. Break-even point calculation + graphic display.....	59
A6. Function for Calculating Linear Depreciation.....	59
A7. Function for Calculating Net Salary.....	60
A8. Class for Calculating Price with VAT.....	60
A9. Implementing Builder for SRL type companies.....	61
A10. Implementing Strategy Pattern for "Good Inferior".....	62

1. Introduction

This documentation describes the development, functionality and use of the MiniContaBill application, developed within the Faculty of Automation and Computers of the "Gheorghe Asachi" Technical University of Iași. MiniContaBill is a desktop application designed to manage accounting and administrative activities for small and medium-sized companies, offering a wide range of functionalities such as: asset depreciation calculation, production cost and profit estimation, supply and demand analysis, as well as employee and salary management.

The project was developed in the C# language, using the .NET Framework 4.8 platform, with an intuitive and modular interface. The structure of the application allows easy use by people with basic economic knowledge, without requiring advanced IT experience.

The purpose of this documentation is to provide a complete picture of the entire application development and usage process, being useful both for developers and testers, as well as for teachers, evaluators or potential users interested in the application's functionality and performance.

2. Software Requirements Specifications

1. Introduction

1.1 Purpose

MiniContaBill is an accounting application designed for companies or business people that want to analyze and manage their financial and personnel data efficiently. The application offers tools for calculating capital depreciation, estimating production costs and profits, analyzing supply and demand through elasticity, as well as managing employees and salaries.

1.2 Document Conventions

This document follows the IEEE standard formatting for software development. The standard defines a regular formatting this document follows including writing to be done in third-person, passive voice as well as readable and grammatically correct text.

1.3 Intended Audience and Reading Suggestions

This part of the document is not intended for the end user because it provides a detailed specification of how the software is to be implemented. Since a user needs information on how

to use the application, instead of how to make one, this document is more geared towards testers and mostly the developers of the application.

The document starts off with an overview of the functions and specifications for this app in section 2, then moves on to describe requirements for interfacing with external hardware and software in section 3. Section 4 describes the app functions in great detail and section 5 lists the requirements the app must adhere to after completion. It is suggested that all audiences of this document start with section 2 first to get a general idea of the software requirements. Testers should next read sections 5.1 through 5.4 (performance, safety, security requirements and software quality attributes). This is to get an idea of how the app will affect them and the system they are running it on, as well as the aspirations for quality. Next a tester should read section 3.1 (user interfaces) followed by all of section 4 (system features). Reading the document in this order will give the tester an idea of what to expect in the interface at first glance, and then they may test all the individual functions to make sure they adhere to the specifications.

After reading section 2, developers should read the remaining sections in order because this document was designed specifically for the purpose of developing the app. Then, how it needs to interface with everything else in section 3 (so they have an idea of what tools to use and how they should use them). Section 4 is the most important to a developer because it describes all the functions of the app in great detail and it will help with making decisions in writing the actual code. Section 5 is considered least important but the developer should still read it to make sure their app is adhered to the given ideals.

1.4Product Scope

MiniContaBill is an accounting application developed in C# using the .NET Framework 4.8. It features a user-friendly interface designed to assist business users in efficiently managing both financial and personnel data. The application is intended to run on any computer that supports C# applications and is optimized for systems with average processing power. MiniContaBill provides functionalities such as capital depreciation calculator, cost and profit estimation, demand and supply analysis, as well as complete employee and salary management.

1.5References

N/A

2. Overall Description

2.1Product Perspective

MiniContaBill is a stand-alone desktop application, intended for business users who want to efficiently manage financial data and personnel information. The application is not part of a larger system and does not require connection to external platforms to perform its main functionality.

Employee data is stored locally, in a JSON file, allowing them to be saved, loaded, modified and deleted without the need for a relational database. This choice offers simplicity, portability and ease of file management, being suitable for users who want a fast and efficient solution without complex dependencies.

The application is developed in the C# language, using the .NET 4.8 framework, which makes it compatible with Windows operating systems that have this or a higher version of the .NET Framework. The interface is organized into thematic tabs (capital, cost and profit, supply and demand, salaries and employees), each with dedicated and intuitive functionalities.

2.2 Product Functions

The MiniContaBill application is structured in four main modules, each with a specific set of functionalities:

Capital Module

- Calculates capital depreciation using three methods: linear, progressive and degressive.
- Displays, for each method, the annual depreciation values, the remaining value and a comparative graph.
- Generates a table with the evolution of the initial and residual value over the years.

Production Cost and Profit Module

- Calculates the final price of the product.
- Determines the profit mass, profit rate and break-even point.
- Calculates the profit tax and the final price including VAT.
- Allows the user to enter the relevant data to obtain these results.

Demand and Supply Module

- Calculates the elasticity of demand according to price or income.
- Calculates the elasticity of supply according to the percentage changes in quantity and price.
- Allows comparative analysis between different scenarios entered by the user.

Salary and Employees Module

- Allows loading and saving the employee list in JSON format.
- Provides functionality for adding, modifying and deleting an employee.
- Calculates the net salary for each employee, the average salary per company and the total salaries.
- Estimates the total cost of the employer, including related contributions.

2.3 User Classes and Characteristics

The MiniContaBill application is intended for business users who want to efficiently manage the financial and administrative aspects of a small or medium-sized company. Currently, the system is designed to be used by a single type of user, without differentiation of roles or permissions.

Main user class: Business Administrator / Accountant

Description: Person responsible for the financial management of the business, who uses the application to calculate depreciation, determine profit and manage employees.

Experience level: No advanced technical experience is required. A basic level of computer use and understanding of elementary economic concepts is assumed.

Needs and objectives: Quick access to automatic accounting calculations, cost and profit estimation, employee management and basic economic analysis.

The application is designed to be intuitive and easy to use, so that users can operate the functionalities without requiring specialized training.

2.4 Operating Environment

The MiniContaBill application is designed exclusively for Windows operating systems, being developed in C# on the .NET Framework 4.8 platform. It is compatible with Windows 10, Windows 11 and other versions that support .NET Framework 4.8 or higher. In order to function properly, the prior installation of this framework is required. The user must have write permissions in the directory where the application saves data (for example, the JSON file for employee management). In addition, a functional keyboard and mouse are required to interact with the interface.

The application is not compatible with Linux or macOS operating systems and cannot be run natively on them. For use on such platforms, it would be necessary to emulate a Windows

environment or run in a properly configured virtual machine. At the same time, the application is not intended for mobile devices (Android/iOS) or web platforms.

2.5 Design and Implementation Constraints

Regulatory Policies: The application must be compliant with GDPR for the protection of data, especially concerning sensitive data in the Salary and Employees Module and customer financial data. As such, the application is standalone and cannot connect to the internet. The application must also comply with IFRS (International Financial Reporting Standards) for using the appropriate calculations.

Hardware Limitations: The application must be supported in low-resource environments, such as low-powered desktops or laptops commonly used in small businesses. This means that the software does not require much processing power or memory usage.

Design Conventions or Programming Standards: The application follows best practices for software maintainability, which includes detailed code documentation and unit tests. Additionally, the codebase is structured in a way that allows for future extensions, such as integrating new financial systems.

2.6 User Documentation

The software includes a Help button integrated within the user interface. When users click this button, it activates a compiled HTML Help (CHM) file, which contains detailed instructions and explanations for features within the application.

The CHM file will be structured with the following sections:

- **Overview:** General information about the application and its core functionalities.
- **Step-by- Step Guides:** Detailed instructions on how to use different features.
- **Troubleshooting:** Guidance for users on resolving potential issues they might encounter, such as errors.

2.7 Assumptions and Dependencies

Use of .NET Framework 4.8

The application is being developed with .NET Framework 4.8, which is expected to be accessible and correctly set up in the intended environment. The development environment and production systems are expected to be compatible with .NET 4.8, along with all required dependencies and runtime libraries.

External DLLs and Third-Party Libraries

The application depends on multiple external DLLs and third-party libraries for user interface rendering and extra features. It is presumed that these libraries are compatible with .NET Framework 4.8 and can be utilized for the duration of the application's lifecycle. Takes for

granted that the licenses for all external components are valid and do not expire. The project team believes that the libraries are free of concealed licensing fees or limitations that might hinder development or delivery. Assumes that all external libraries or DLLs integrated into the project will remain maintained and supported during the development process. Should any of these libraries become deprecated or unsupported during the project, there may be a need to find alternatives or reimplement parts of the functionality.

3. External Interface Requirements

3.1 User Interfaces

General Structure: The application has a single main window that serves as the container for different functional panels. The window layout follows a tabbed interface or panel container system where users can switch between different functionalities without opening new windows. Each panel represents a distinct feature or functionality, such as firm name and type, firm attributes, or salary management, and is displayed within a dedicated area of the main window. All of which is not resizable.

Panel Design: Panels will include standard UI controls (e.g., buttons, textboxes, dropdowns) that align with the functionality they represent. The design will follow flat UI elements and system fonts that are legible and scalable across different devices and screen resolutions

Standard UI Elements: A universal Help button will be included on the navigation bar and accessible from every panel. Clicking this button will open the CHM-based help file. A fixed navigation panel (either horizontally at the top and/or vertically on the left) will be used for switching between different major sections of the application.

3.2 Hardware Interfaces

N/A

3.3 Software Interfaces

Operating Systems: The application is built using .NET Framework 4.8, which is compatible with Windows operating systems and will be deployed on Windows computers. The application communicates with the Windows Operating System to interact with system resources and manage files (e.g. saving the employee list in JSON format in a file).

External Libraries:

- **RealTailor (Version 3.8.1.2):** Used for user controls to make the applications more professional looking

3.4 Communications Interfaces

N/A

4. System Features

In this section, the functional requirements are organized into the main modules of the application: Capital, Production Cost and Profit, Demand and Supply, and Salaries and Employees. This structuring reflects the application interface and the functional logic of the system.

4.1 Capital Module

4.1.1 Description and Priority

The "Capital" module allows the user to calculate the depreciation of the value of an asset over its economic life. The user can select the desired depreciation method (linear, progressive or declining balance), and the system automatically calculates the annual depreciated value, the residual value and generates a graph and a table with their evolution.

Priority: High

Detailed priority rating:

Benefit: 9 (essential for asset-related financial decisions)

Penalty: 7 (lack of this feature would make the application incomplete for accounting management)

Cost: 4 (medium complexity of implementation)

Risk: 3 (calculation is stable and easy to verify, with low risk of error)

4.1.2 Stimulus/Response Sequences

Purpose: The user enters data about an asset and chooses the depreciation method, and the system calculates the annual values and displays the results in a table and a graph.

Steps (sequences of user actions and system responses):

- The user opens the "Capital" tab in the application interface.
- The system loads and displays the asset data entry form.
- The user enters the initial value of the asset, the life span and the residual value.
- The system validates the entered data (numeric, non-zero, positive, etc.).

- The user selects the depreciation method: straight-line, progressive or declining balance.
- The system highlights the selected method and activates the "Calculate" button.
- The user clicks the "Calculate" button.
- The system processes the data and calculates the depreciated value for each year.
- The system displays a table with the annual depreciation values and the residual value.
- In parallel, the system generates a graph illustrating the decrease in the asset's value over time.

4.1.3 Functional Requirements

REQ-CAP-1: The application must allow the user to enter the initial value of the asset, the useful life (in years) and the residual value.

REQ-CAP-2: The application must validate that the initial value and useful life are positive numbers. If the user enters invalid values (e.g. text, zero or negative numbers), the system automatically puts the zero value.

REQ-CAP-3: The user must be able to select one of the three depreciation methods: straight-line, progressive, declining balance.

REQ-CAP-4: After pressing the "Calculate" button, the application must automatically calculate the annual depreciated value and the residual value according to the selected method.

REQ-CAP-5: The application must display the calculation results in a table, with a row for each year and columns for: the current year, the depreciated value in that year and the remaining value.

REQ-CAP-6: The application must generate a graph that visually illustrates the evolution of the asset value over time.

REQ-CAP-7: If one of the fields is left blank, the system must warn the user and not allow the calculation to run.

4.2 Production Cost and Profit Module

4.2.1 Description and Priority

The Production Cost and Profit module allows the user to estimate the financial viability of a business idea or production process by calculating key indicators such as profit margin, profitability rate, breakeven point, final price with and without VAT, and profit tax. The user inputs production cost and desired profit margin, and the system computes the relevant financial metrics.

Priority: High

Detailed priority rating:

Benefit: 9 (critical for pricing decisions and business profitability analysis)

Penalty: 8 (without this module, the user would not be able to determine breakeven or final price, severely limiting decision-making)

Cost: 4 (medium complexity; uses straightforward financial formulas)

Risk: 4 (relatively low; risk lies mainly in rounding errors or incorrect input validation)

4.2.2 Stimulus/Response Sequences

Purpose:

The user inputs the unit cost of production, desired profit margin, and applicable tax rates. The system calculates and displays the final price (with and without VAT), profit, profit rate, breakeven point, and profit tax.

Steps:

- The user opens the Production Cost and Profit tab.
- The system loads and displays the input form for cost-related data.
- The user enters:
 - Unit production cost
 - Desired profit margin (%)
 - Fixed costs (if applicable)
 - Number of units produced
 - VAT rate (%)
 - Profit tax rate (%)
- The system validates that all numerical inputs are positive and not empty.
- The user clicks the "Calculate" button.
- The system calculates:
 - Profit
 - Profit rate
 - Final price without VAT
 - Final price with VAT
 - Profit tax
 - Breakeven point
- The system displays the results in a textBox.

4.2.3 Functional Requirements

REQ-CPP-1: The application must allow the user to enter the following data: raw material cost, labor cost, indirect costs and product selling price.

REQ-CPP-2: The application must validate that all values entered are positive numbers. If the user enters invalid values (e.g. text, zero or negative numbers), the system must display a clear error message and not allow the calculations to continue.

REQ-CPP-3: After pressing the "Calculate" button, the application must automatically calculate the total production cost, the profit rate, final price, breakeven point, price with VAT.

REQ-CPP-4: The application must display the results in a textBox

REQ-CPP-5: If one of the required fields is left blank, the system must warn the user and not allow the calculations to be performed.

4.3 Demand and Supply Module

4.3.1 Description and Priority

The Demand and Supply module is a key component of the application, aimed at analyzing how product demand and supply respond to changes in price and income. This module is essential for understanding market behavior and making informed business decisions.

Priority: High, as it provides core insights into price sensitivity and consumer/producer behavior.

Detailed Priority Rating:

Benefit: 9 – The module is essential for evaluating pricing strategies and forecasting economic outcomes.

Penalty: 8 – Without this functionality, users would be unable to determine breakeven points or optimal pricing, which significantly hampers decision-making.

Cost: 4 – The implementation involves medium complexity, relying on established financial formulas and standard input validation.

Risk: 4 – The risk is relatively low, with potential issues primarily related to rounding errors or invalid input data.

4.3.2 Stimulus/Response Sequences

Purpose:

The user inputs the initial and new values of price, quantity demanded, and quantity supplied, as well as the change in income if analyzing income elasticity. The system calculates and displays the price elasticity of demand, price elasticity of supply, and income elasticity of demand or income elasticity of supply.

Steps:

- The user opens the Demand and Supply tab.
- The user enters the following values:
 - Initial price
 - New price

- Initial income
- New income
- Initial quantity demanded
- New quantity demanded
- Initial quantity supplied
- New quantity supplied
- The system validates that all numerical inputs are positive and not left empty.
- The user clicks the "Calculate" button.
- The system performs the following calculations:
 - Price Elasticity of Demand
 - Price Elasticity of Supply
 - Income Elasticity of Demand
 - Income Elasticity of Supply
- The system displays the calculated elasticities in a textBox, along with interpretations of whether demand/supply is elastic, inelastic, or unitary.

4.3.3 Functional Requirements

REQ-DS-1: The application must allow the user to input:

- Initial price
- New price
- Initial income
- New income
- Initial quantity demanded
- New quantity demanded
- Initial quantity supplied
- New quantity supplied

REQ-DS-2: The application must validate that all numerical inputs are positive and not empty. If invalid data is entered (e.g., text, negative or zero values), the system must display a clear error message and disable calculation.

REQ-DS-3: The user must be able to select which type of elasticity they want to calculate:

- Price elasticity of demand
- Price elasticity of supply
- Income elasticity of demand
- Income elasticity of supply

REQ-DS-4: When the "Calculate" button is pressed, the system must compute the selected elasticities using the appropriate formulas:

- Price Elasticity of Demand
- Price Elasticity of Supply
- Income Elasticity of Demand

- Income Elasticity of Supply

REQ-DS-5: The application must display the results in a textBox.

REQ-DS-6: If a required field is missing or contains invalid input, the application must prevent the calculation and inform the user via an error message.

4.4 Salaries and Employees Module

4.4.1 Description and Priority

The Salary and Employees module manages employee data for the selected company and allows full CRUD operations (Create, Read, Update, Delete). It enables users to load and display employee information from a JSON file, add new employees, edit existing ones, or remove them. Additionally, it offers financial calculations such as average salary, total salaries, net salary per employee, and total employer cost.

Priority: High

Detailed priority rating:

- **Benefit:** 9 (essential for basic HR and salary accounting functionalities)
- **Penalty:** 8 (without it, the app would lack personnel cost tracking, affecting financial accuracy)
- **Cost:** 5 (moderate implementation complexity due to CRUD logic and multiple calculations)
- **Risk:** 4 (moderate; potential errors in salary computations if data is incomplete or inconsistent)

4.4.2 Stimulus/Response Sequences

Purpose:

The user can manage the list of employees for a selected company directly in a DataGridView and trigger specific financial calculations using dedicated buttons.

Steps:

1. The user selects a company from the interface.
2. The system searches the local folder for a corresponding employee file (e.g., a .json file named after the company).
 - If the file exists, the system automatically loads the data and populates the DataGridView with the list of employees.

- If the file does not exist, the system displays an empty DataGrid, allowing the user to begin adding employee data.
- 3. The user can add a new employee by inserting a new row directly in the DataGrid and completing the required fields (e.g., id, name, gross salary, role).
- 4. The user can modify existing data by directly editing the cells of the DataGrid.
- 5. The user can delete an employee by selecting the row and pressing the Delete button.
- 6. The user can initiate specific financial calculations by pressing the corresponding buttons:
 - **Calculate Net Salary:** calculates the net salary for each employee.
 - **Average Salary:** calculates and displays the average gross salary.
 - **Total Salaries:** sums up all gross salaries.
 - **Total Employer Cost:** calculates the total employer cost, including taxes and contributions.
- 7. Before performing any calculations, the system validates the input values in the DataGrid:
 - Salaries must be numeric and positive.
 - Fields must not be left blank.
 - In case of invalid input, the system highlights the issue and displays an error message.
- 8. When modifications are complete, the user can save the updated employee list to the associated JSON file for the selected company.

4.4.3 Functional Requirements

REQ-SAL-1: The system must allow the user to enter the company of a company

REQ-SAL-2: Upon selecting a company, the system must search the local directory for a corresponding .json file containing employee data and load it into the DataGrid, if it exists.

REQ-SAL-3: If the employee file does not exist for the selected company, the system alerts the user that there is no employee file for that company and displays an empty DataGrid and allows the user to input new employee data.

REQ-SAL-4: The user must be able to add new employees by filling a textBox for each column in the dataGrid.

REQ-SAL-5: The user must be able to modify existing employee data by editing cells in the DataGrid.

REQ-SAL-6: The user must be able to delete an employee by selecting a row and clicking a Delete button.

REQ-SAL-7: The system must validate that salary fields contain numeric and positive values, and that no mandatory field is left empty. Invalid inputs must be flagged with an error message.

REQ-SAL-8: The system must calculate and display the net salary for each employee when the Calculate Net Salary button is pressed.

REQ-SAL-9: The system must calculate and display the average gross salary across all employees when the Average Salary button is pressed.

REQ-SAL-10: The system must calculate the total of all gross salaries when the Total Salaries button is pressed.

REQ-SAL-11: The system must calculate the total employer cost (gross salaries + employer contributions) when the Total Employer Cost button is pressed.

REQ-SAL-12: The user must be able to save the updated DataGrid content to the .json file associated with the selected company.

REQ-SAL-13: All calculations must be updated in real-time when triggered, without the need to reload the application.

Other Nonfunctional Requirements

4.5 Performance Requirements

The MiniContaBill application is designed to function efficiently on general-purpose personal computers and office workstations, without the need for high-performance hardware. The following performance criteria must be met to ensure a smooth and responsive user experience:

- **Startup Time:** The application should launch and become responsive within **a few seconds** of execution under normal operating conditions.
- **Data Loading Time:** Loading employee data from a local JSON file should complete in **under one second** for typical use cases involving **hundreds of records**.
- **Calculation Time:** Financial computations (e.g., depreciation, salary statistics, profit estimations) should complete **almost instantly**, generally within **half a second**, for standard input sizes.
- **User Interface Responsiveness:** All interface elements (tabs, buttons, data entry fields, grids) must respond to user actions with **minimal delay**, typically **under 100**

milliseconds, to maintain usability and interactivity.

- **Resource Usage:** The application should use system resources efficiently and remain stable during use. Memory and CPU usage should remain moderate to ensure compatibility with most office environments, without significantly impacting overall system performance.
- **File Handling:** Reading from and writing to local storage (e.g., employee JSON files) must be performed quickly, without noticeable delay for files of moderate size.

These performance goals ensure that MiniContaBill remains practical and usable for small to medium-sized businesses operating in typical desktop environments.

4.6 Safety Requirements

The MiniContaBill application is a desktop-based financial and personnel management tool that does not interact with physical hardware or critical safety systems. Nevertheless, due to the nature of the data it processes (financial calculations and employee information), certain precautions must be considered to minimize the risk of data loss or user error.

Data Integrity Protection:

The application performs basic input validation to avoid incorrect or nonsensical financial computations (e.g., preventing the use of negative numbers or missing fields).

Error Handling:

The application handles invalid inputs with clear error messages, blocking further calculations until valid data is provided. This prevents propagation of incorrect values into financial outputs.

File Access Handling:

In the event that a file is missing, corrupted, or improperly formatted, the system avoids crashing and informs the user with a relevant message. However, advanced file protection or versioning is not currently implemented.

Compliance and Data Protection:

While not subject to official financial certification, if used in a real company environment, care must be taken to store personal data in accordance with local data protection laws (e.g., GDPR in the EU). No sensitive data is encrypted by default in the current version.

Safety Certification:

No safety certifications are required for this product. However, adherence to best practices in user input validation, data separation, and structured file handling is followed in the current implementation.

4.7 Security Requirements

MiniContaBill does not implement a data authentication or encryption system in the current version. All financial and employee information is saved locally in JSON files, in text format. The user is responsible for protecting these files at the operating system level. For future versions, the following security improvements are being considered:

- Password-based authentication;
- Local file encryption;
- Access control and logging of sensitive operations.

The application is not intended for use in regulated environments (e.g. GDPR, ISO 27001), but users are encouraged to avoid storing sensitive personal data in unsecured environments.

4.8 Software Quality Attributes

Usability:

MiniContaBill is designed to be user-friendly, even for non-technical users. The graphical interface is intuitive, structured into tabs that correspond to the application's main features, with clear and direct actions (e.g., buttons for calculations, tables, charts). The interface utilizes the **ReaLTaiZor** library to provide a modern, consistent look and user-friendly controls.

Maintainability:

The application is developed modularly, using external libraries (DLLs) for core calculation functions such as depreciation, salary, and elasticity computations. The UI, partially built with ReaLTaiZor components, can be easily updated or extended without affecting the application's logic layer.

Reusability:

Key computational functions are implemented in reusable DLLs, allowing them to be integrated into other financial or accounting systems with minimal changes.

Portability:

The application runs on any system that supports .NET Framework 4.8. A future version may migrate to .NET Core or .NET 6+ to support cross-platform compatibility (Windows, Linux, macOS).

Correctness and Reliability:

All core algorithms are tested to ensure accurate results. The system validates user input to avoid errors (e.g., empty fields, negative values) and ensures reliable operation across different usage scenarios.

Robustness:

The application handles common errors gracefully, such as missing files, invalid input, or incomplete data. It provides clear error messages and prevents the user from proceeding until issues are resolved.

Flexibility:

The modular structure allows for easy addition of new features (e.g., extra calculation methods, reporting tools, employee categories) without affecting existing functionality.

4.9 Business Rules

The application follows basic access control principles through its usage logic:

- Only after selecting a company can the user perform operations related to employees or salaries
- Employee data is linked to a specific company and is only loaded or saved if the corresponding file exists in the designated folder.
- Users can calculate financial metrics (e.g., depreciation, production costs, profitability) independently, without needing to follow a strict sequential flow.
- Each calculation module is activated only when the required input data is provided and validated.
- Data entry and editing are done directly within the interface (e.g., DataGrid for employees), and changes must be confirmed through action buttons (e.g., "Add", "Delete", "Calculate").
- File operations such as load and save must be done manually by the user.

These principles ensure that the user maintains full control over the data flow and calculations while minimizing the risk of unintended actions.

5. Other Requirements

Internationalization: The current version is localized in Romanian. The interface and messages are displayed in Romanian only. Future versions may consider adding multilingual support depending on user needs.

Third-party Libraries: The UI benefits from the use of the **RealTailor** library for improved visual appearance of buttons and controls.

Appendix A: Glossary

SRS (Software Requirements Specification) – A document that describes all the functional and non-functional requirements for a software product.

JSON (JavaScript Object Notation) – A lightweight data-interchange format used to store employee and company data in this application.

DLL (Dynamic Link Library) – A file that contains reusable functions or classes, used in the application for modularizing financial and mathematical calculations.

UI (User Interface) – The graphical interface through which users interact with the application.

TVA (Taxa pe Valoarea Adăugată) – The Romanian acronym for Value Added Tax (VAT), included in cost and pricing calculations.

Elasticity – An economic measure that shows how the quantity demanded or supplied of a product changes in response to a change in price or income.

Depreciation – The reduction in the value of an asset over time, calculated using different methods (linear, progressive, declining balance).

ReaLTaiizor – A third-party .NET UI library used to enhance the visual design of controls like buttons and forms.

Residual Value – The remaining value of an asset at the end of its useful life.

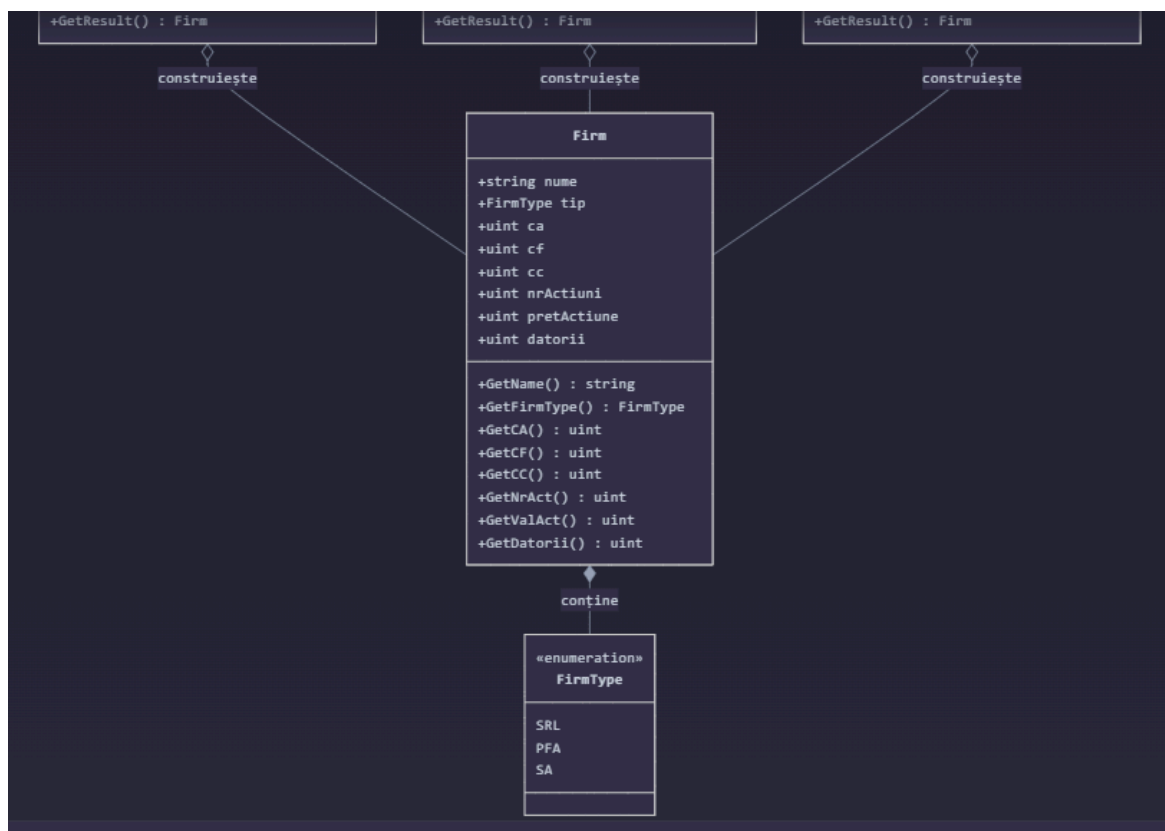
Break-even point – The point at which total cost and total revenue are equal, meaning the business covers all costs but makes no profit.

3. Analysis Models

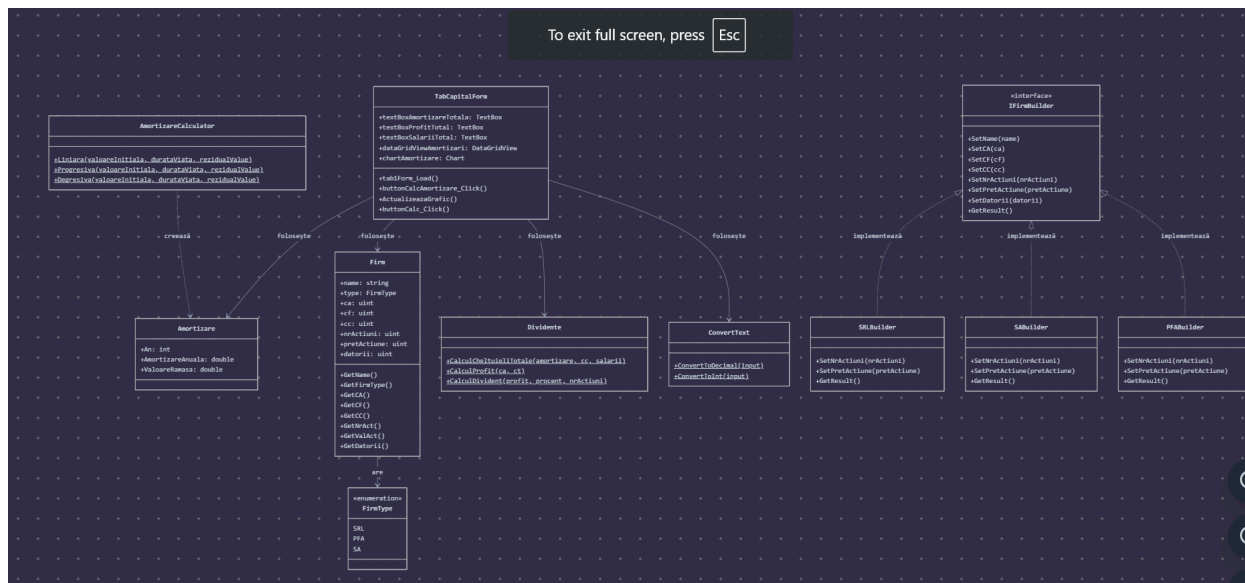
3.1 Class Diagrams

3.1.1 Select Form

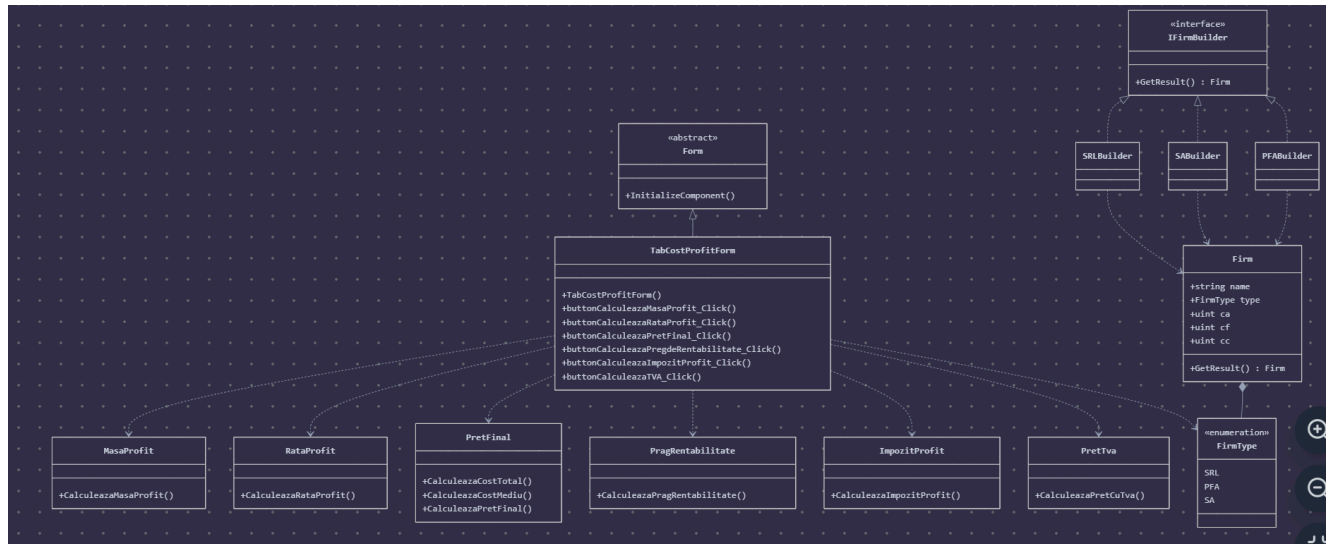




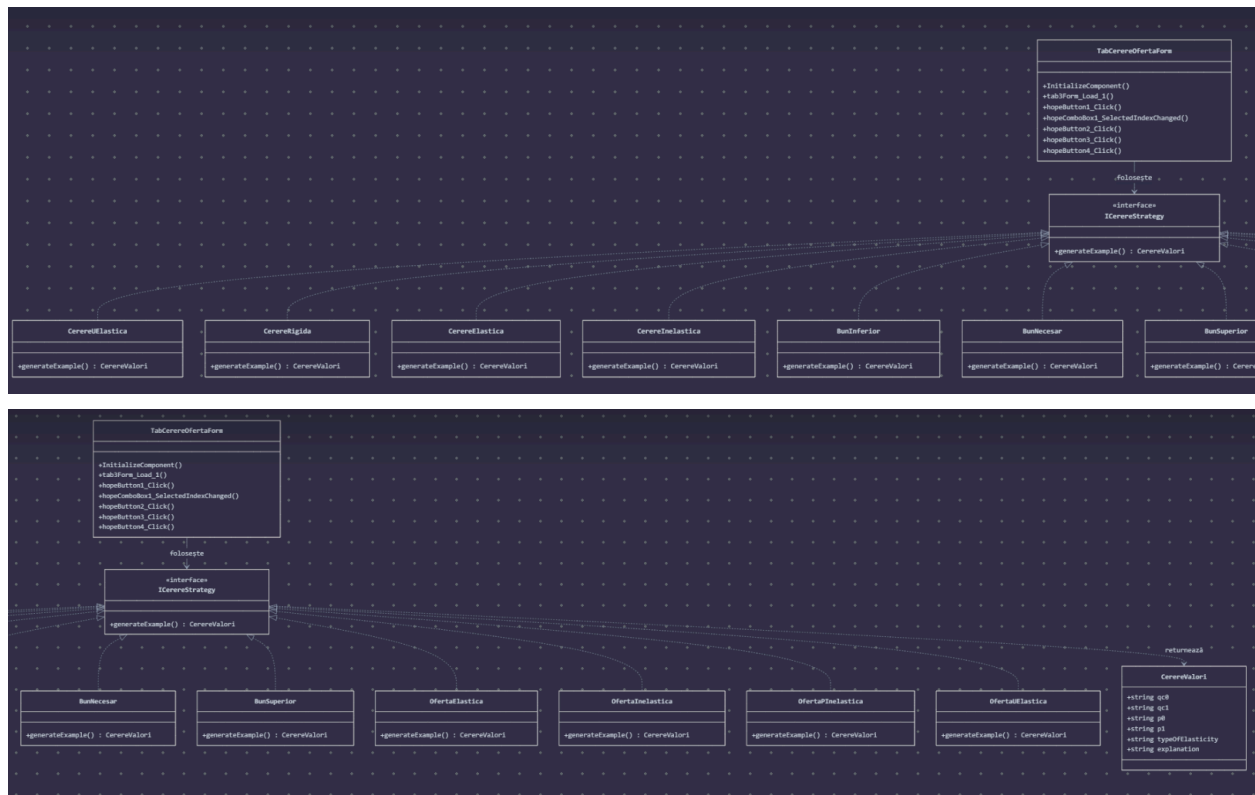
3.1.2 Capital Form



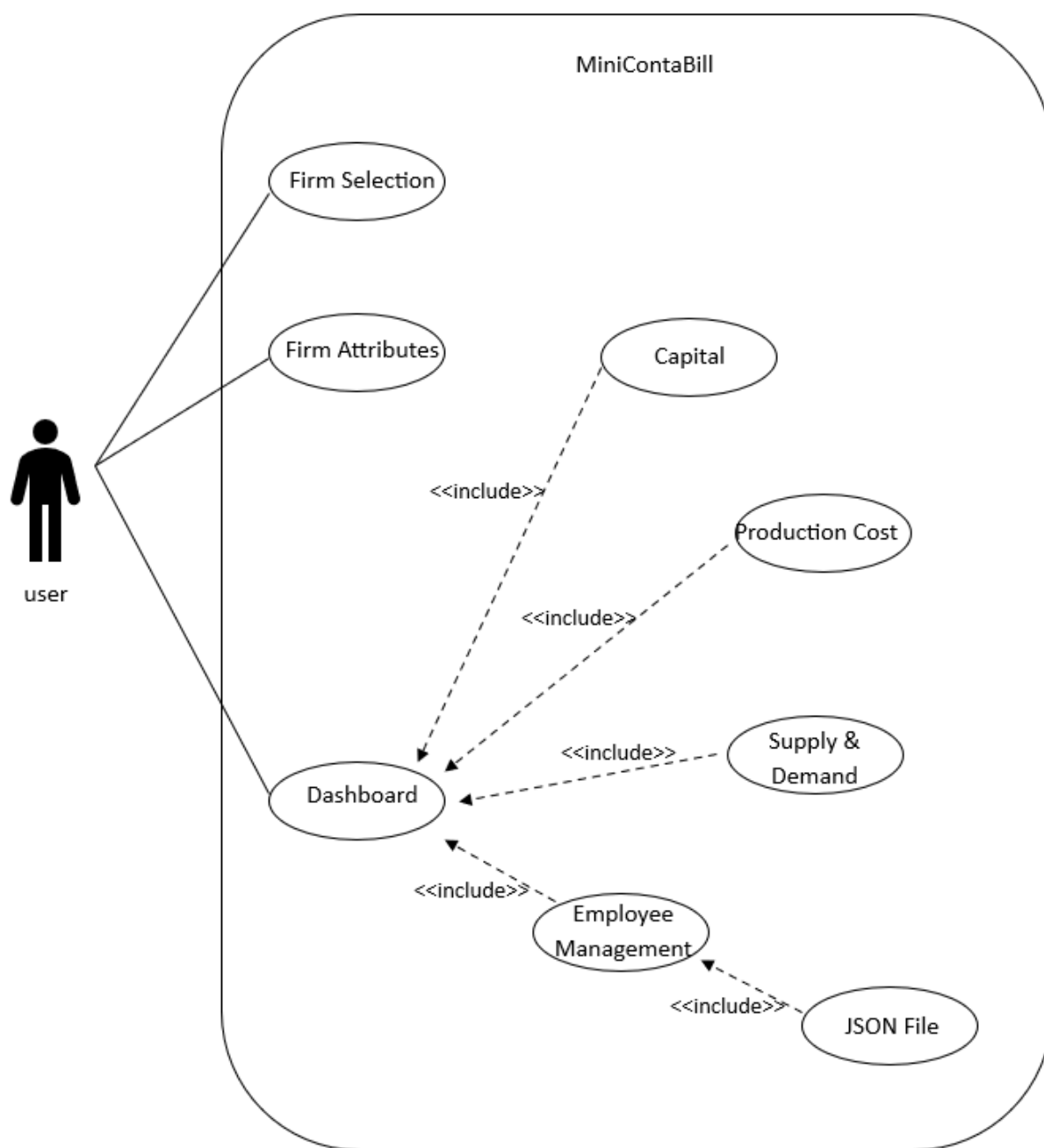
3.1.3 Production Cost and Profit Form



3.1.4 Demand and Supply Form

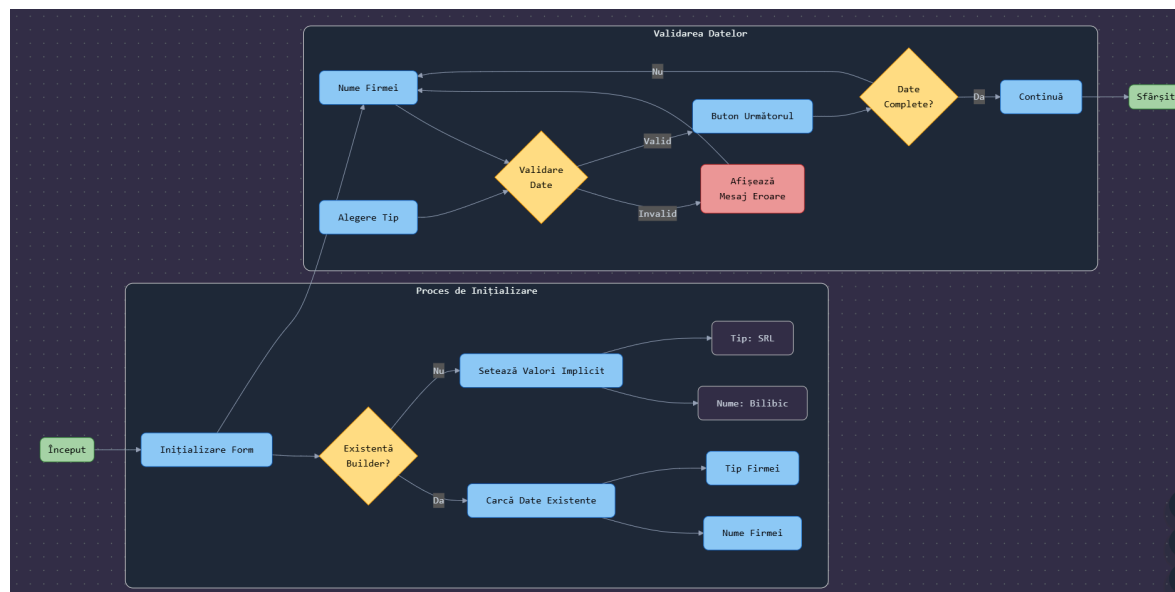


3.2 UseCase Diagrams

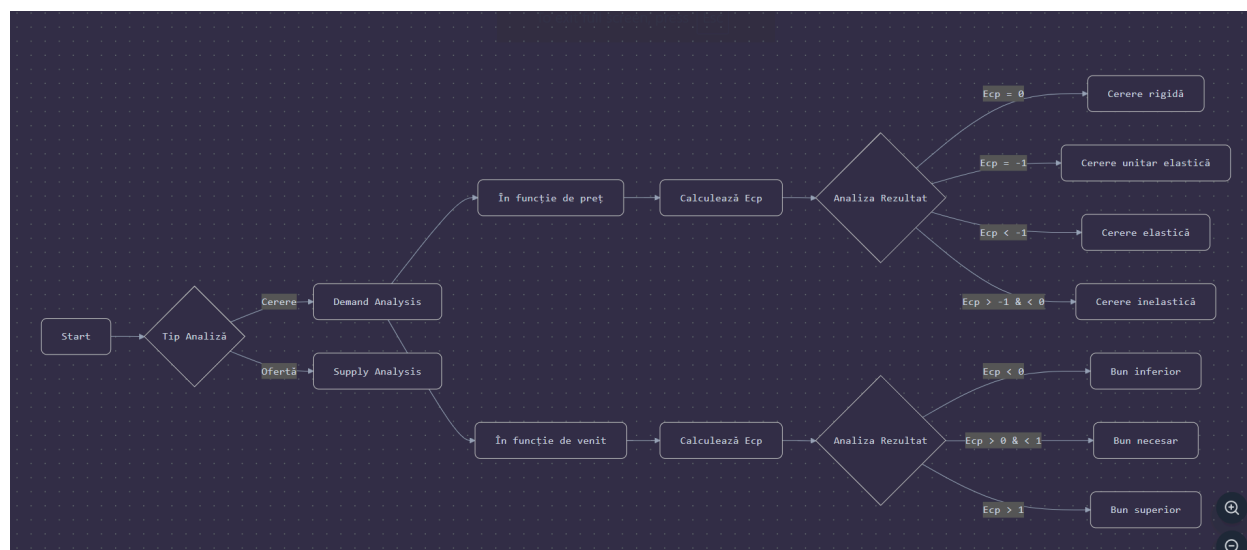


3.3 Activity Diagrams

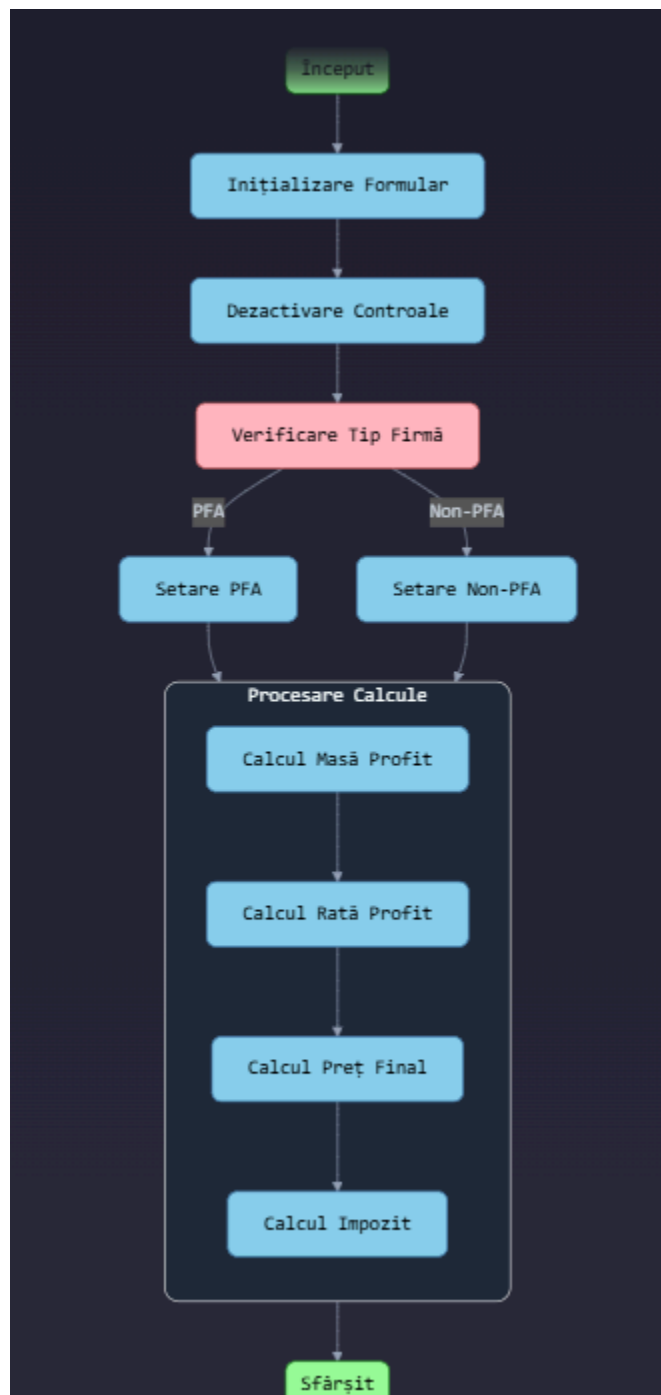
Select Form



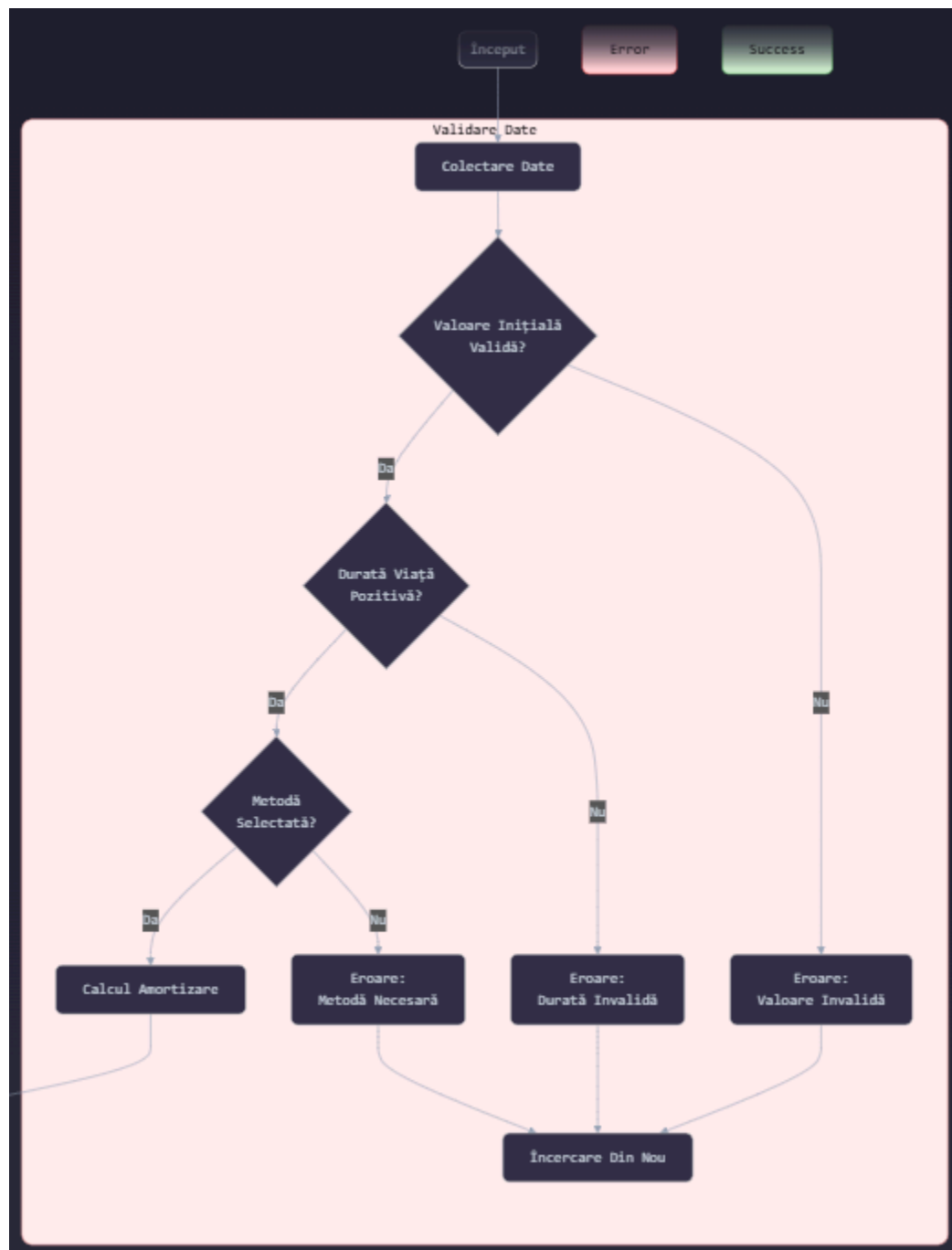
Demand and Supply Form

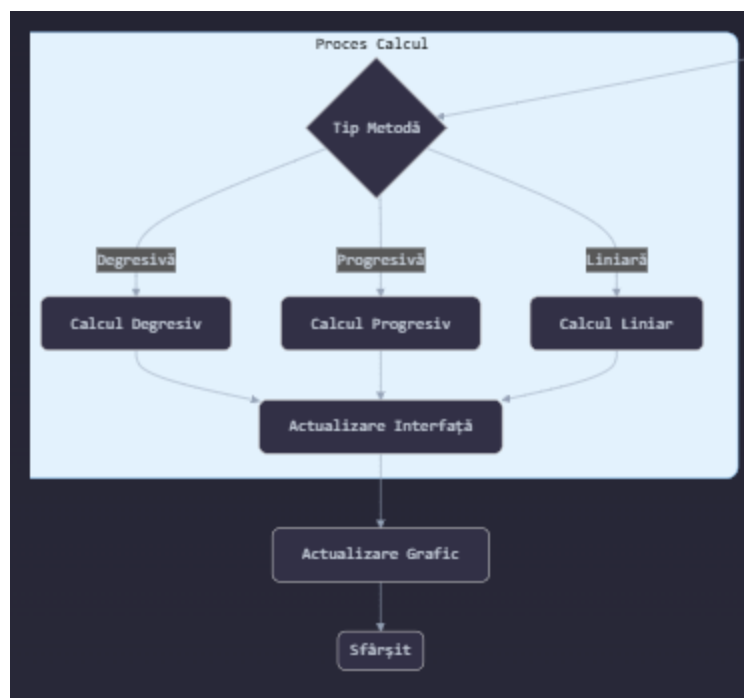


Production Cost and Profit Form

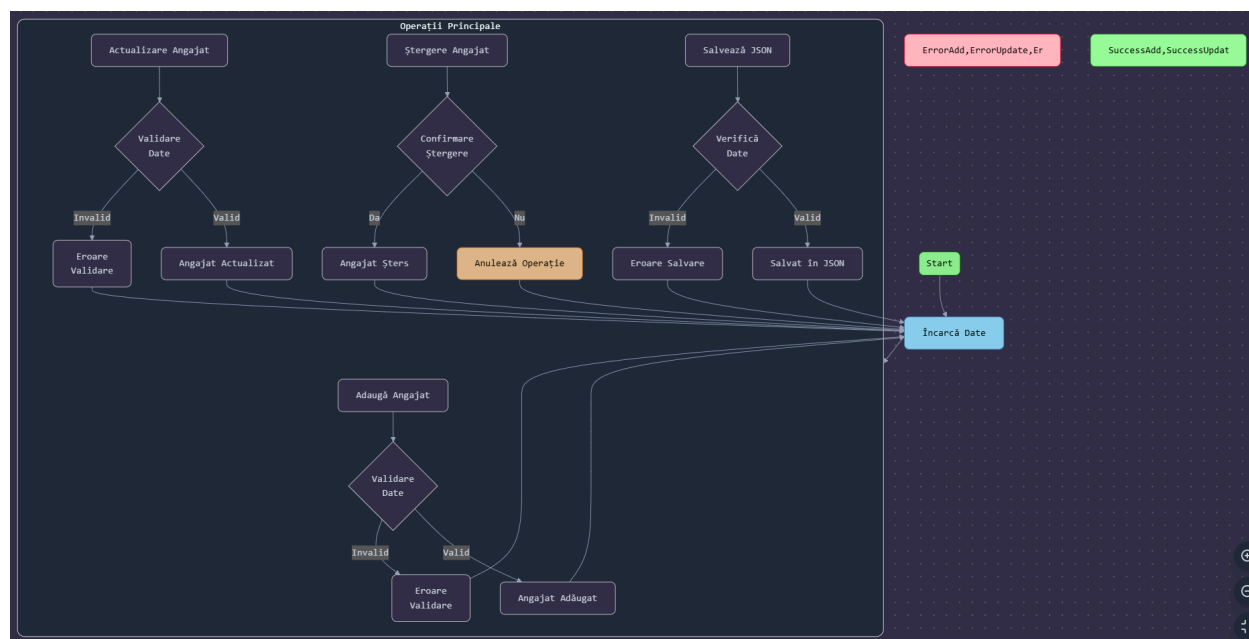


Capital Form



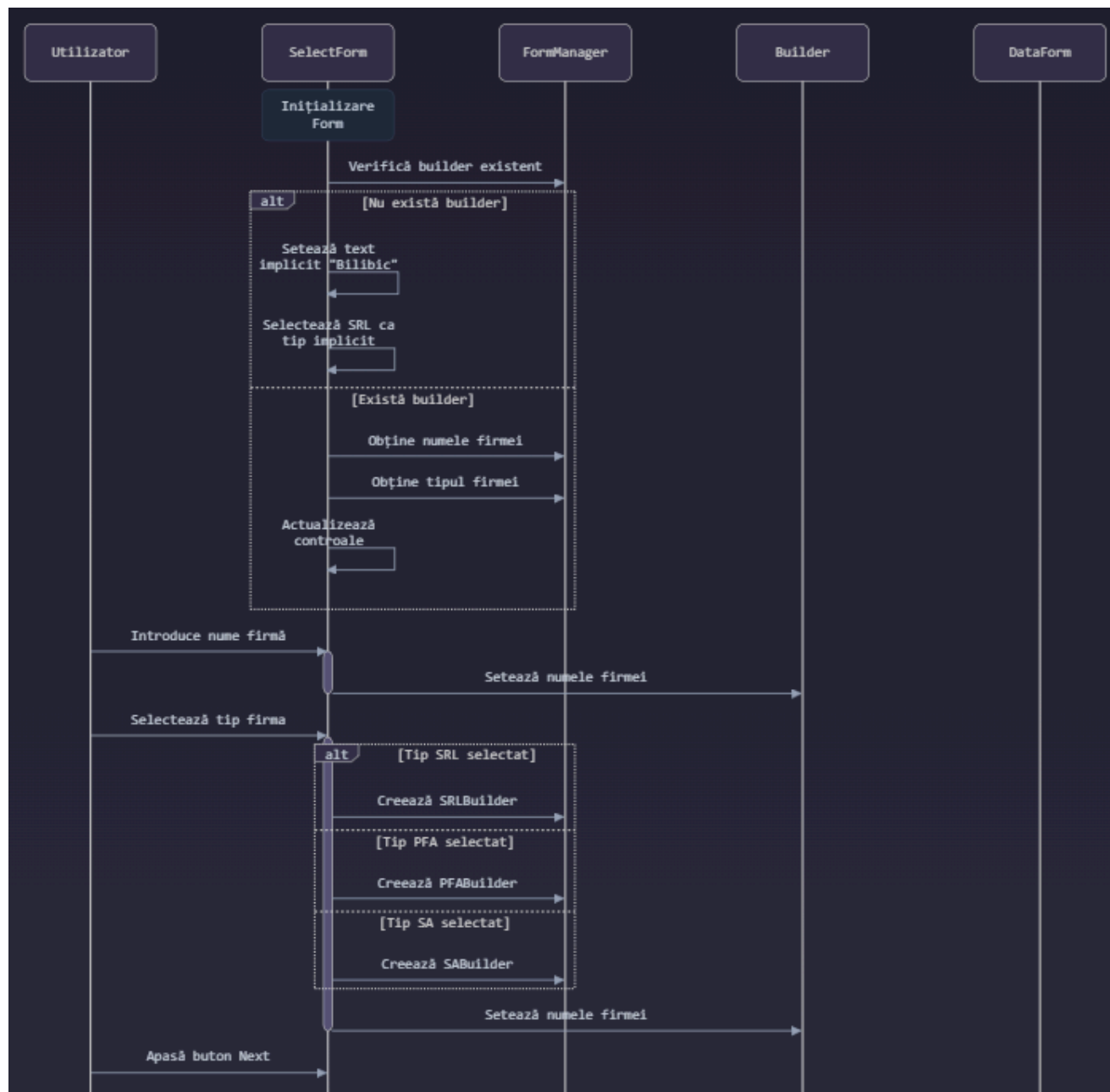


Salaries and Employees Form

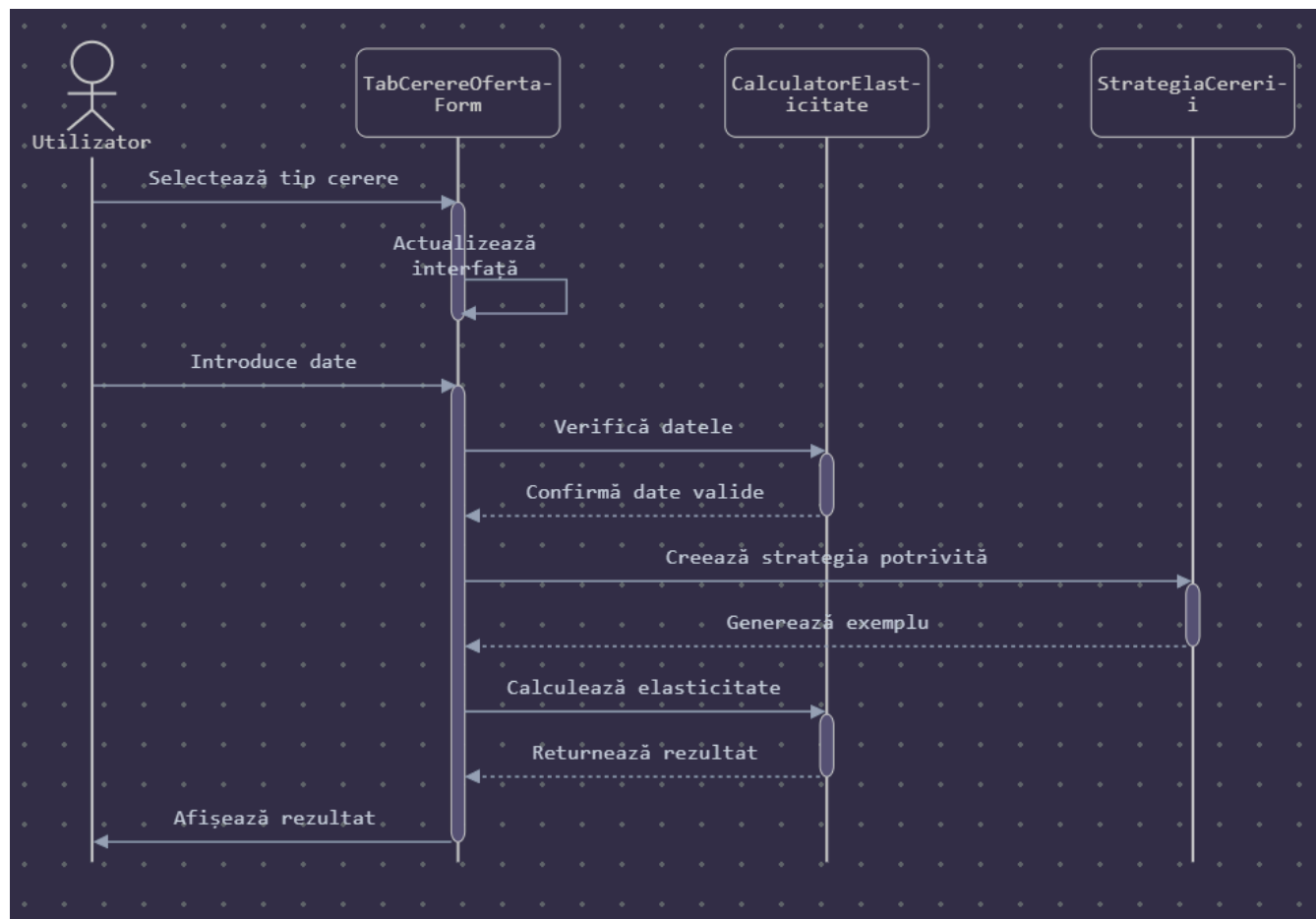


3.4 Sequences Diagrams

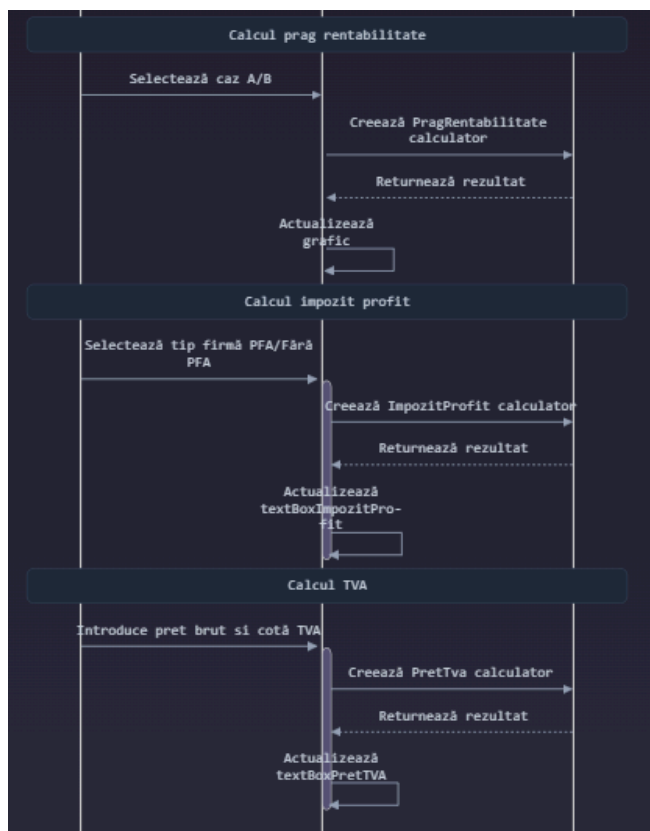
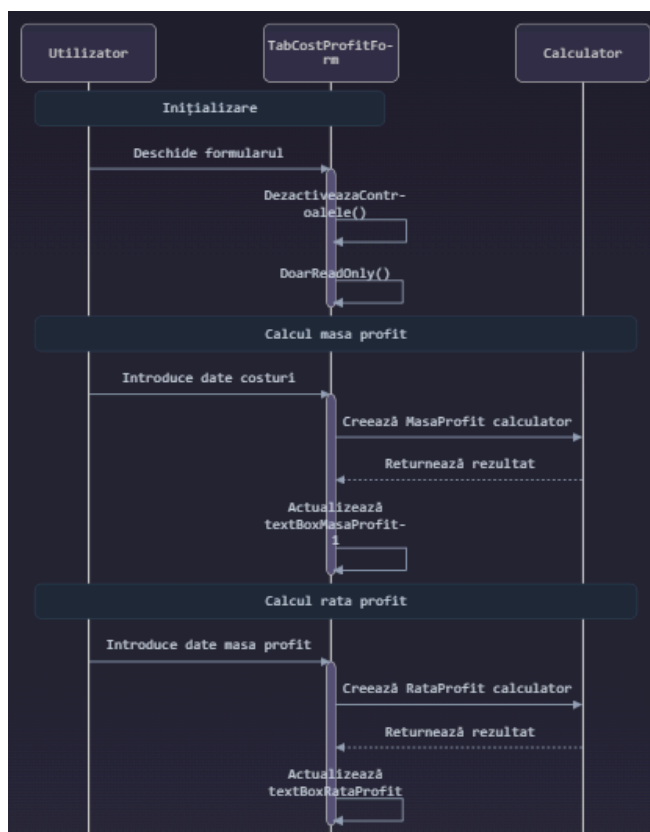
Select Form



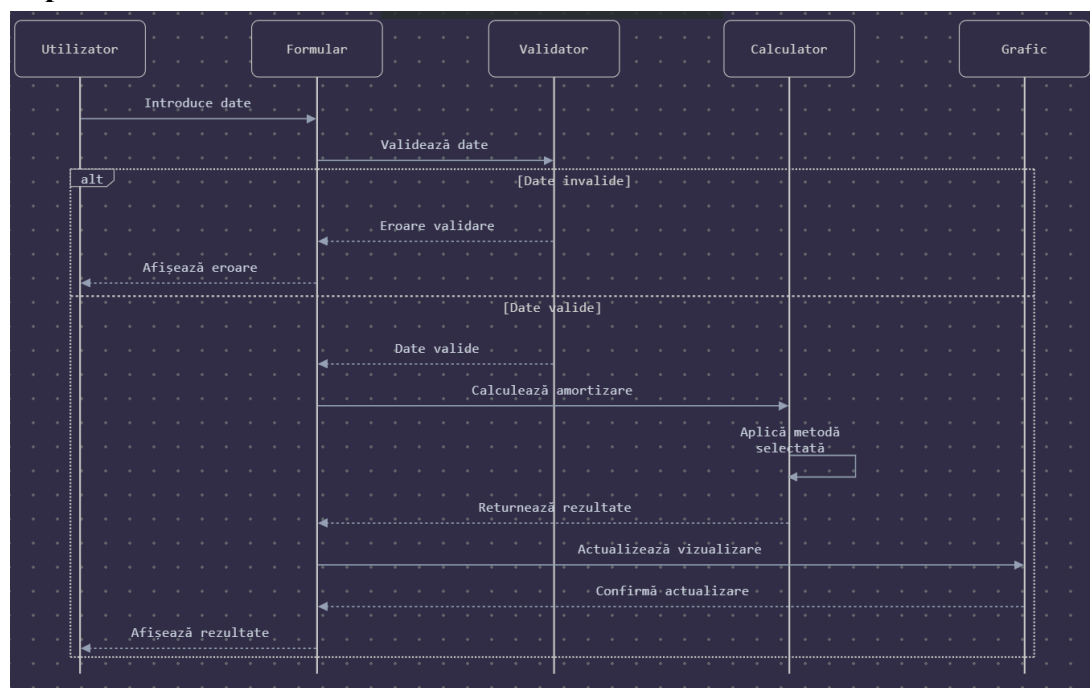
Demand and Supply Form



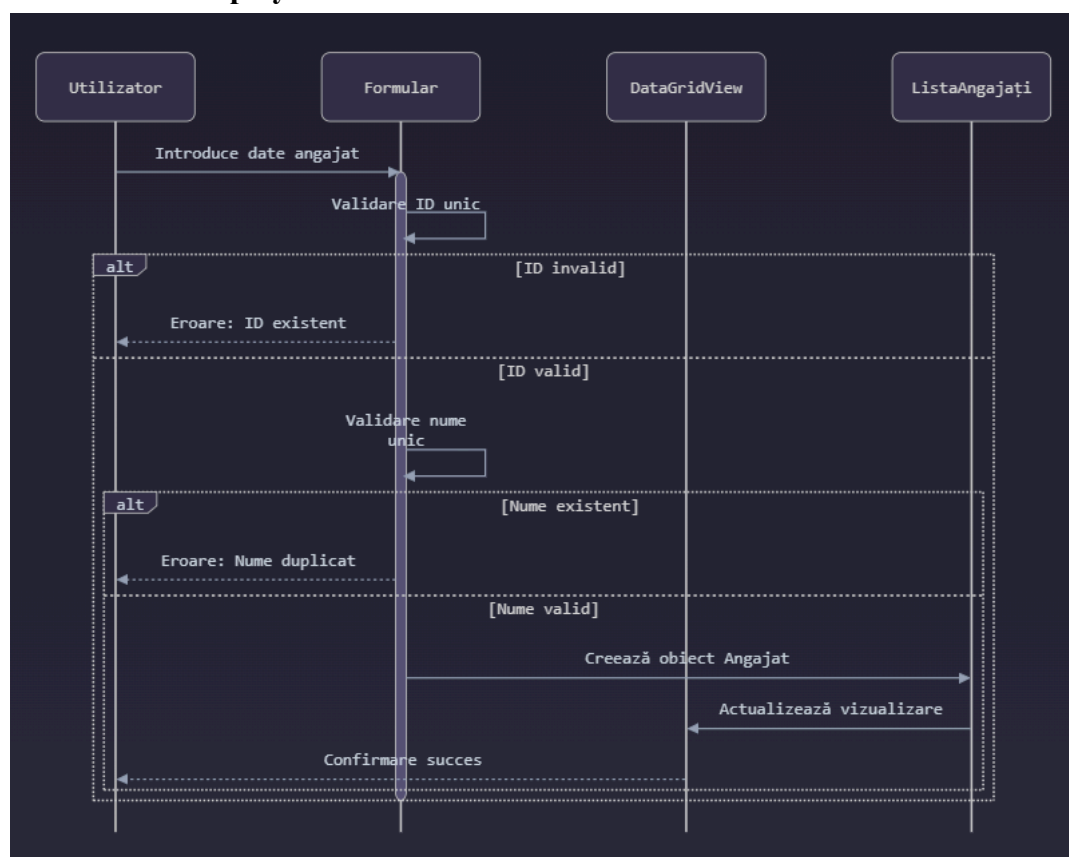
Production Cost and Profit Form



Capital Form



Salaries and Employees Form



4. Design Patterns

4.1 Builder (Company Object Construction)

To manage the attributes of a company — within which various financial calculations can be applied — the **Builder** creational design pattern was used. This pattern allows the construction of a complex object (the Company class) that has multiple attributes and can have different configurations depending on the company type.

Firm Object

The Firm class contains several attributes (e.g., name, CA, CF, debts, number of shares). Some attributes are only applicable to certain firm types — for instance, *shares* are relevant only for SA companies. Default values are initialized upon object creation to prevent unexpected runtime errors.

```

/// <summary> Interfață pentru un builder al unui obiect tip Firm
9 references
public interface IFirmBuilder
{
    /// <summary> Metodă ce setează numele firmei
    6 references | 1/1 passing
    void SetName(string name);

    /// <summary> Metodă ce setează cifra de afaceri
    5 references | 1/1 passing
    void SetCA(uint ca);

    /// <summary> Metodă ce setează capitalul fix
    4 references
    void SetCF(uint cf);

    /// <summary> Metodă ce setează capitalul circulant
    4 references
    void SetCC(uint cc);

    /// <summary> Metodă ce setează număr de acțiuni din care e compus firma
    5 references | 1/1 passing
    void SetNrActiuni(uint nrActiuni);

    /// <summary> Metodă ce setează prețul unei acțiuni
    4 references
    void SetPretActiune(uint pretActiune);

    /// <summary> Metodă ce setează datoriile firmei
    4 references
    void SetDatorii(uint datorii);

    /// <summary> Metodă ce returnează obiectul Firm construit
    20 references | 5/5 passing
    Firm GetResult();
}

```

IFirmBuilder Interface

The IFirmBuilder interface defines the necessary steps to construct a Firm object. The GetResult() method returns the current version of the built object at any given time.

Concrete Builders: SRLBuilder, SABuilder, PFABuilder

These classes implement the IFirmBuilder interface with logic specific to each type of firm.

For example:

- SABuilder sets share-related fields such as number of shares and share price.
- SRLBuilder and PFABuilder set those fields to zero, as they are not applicable for those company types.

```

/// <summary> Metodă ce setează cu 0 numărul de acțiuni
3 references | 1/1 passing
public void SetNrActiuni(uint nrActiuni) => firm.nrActiuni = 0;

/// <summary> Metodă ce setează cu 0 prețul unei acțiuni
2 references
public void SetPretActiune(uint pretActiune) => firm.pretActiune = 0;

```

Each builder method is invoked as needed based on user interaction with the interface, allowing the dynamic creation of a correctly-configured Company object.

Advantages of this approach:

- **Easy extensibility:** To support a new company type, a new builder class can simply be added without modifying existing code.
- **Separation of concerns:** The construction logic is decoupled from the Company class, following the *Single Responsibility Principle*.
- **Reusability and clarity:** Builders can be reused across different contexts, improving code maintainability and readability.

4.2 Strategy

To select the elasticity type examples in a flexible and organized way, the Strategy pattern was used. It allows the choice of a specific behavior (e.g. elastic demand, inelastic demand, inferior good, etc.) without the application interface knowing the logical details behind each case.

RequestValues Object: The struct type object contains the values that will be processed and displayed

```

/// <summary>
/// Obiectul prelucrat și afișat ce conține informațiile.
/// </summary>
56 references
public struct CerereValori
{
    public string qc0;
    public string qc1;
    public string p0;
    public string p1;
    public string typeOfElasticity;
    public string explanation;
}

```

Thus, depending on the desired output, an object is returned that implements the functionality specific to that elasticity type, through a common interface.

Interface ICerereStrategy:

```

/// <summary>
/// Interfața prin care se construiesc obiectele
/// </summary>
22 references
public interface ICerereStrategy
{
    22 references
    CerereValori generateExample();
}

```

Each class corresponds to a type of elasticity of demand or supply and implements the generateExample() method with specific data and explanations. The classes include:

Demand:

- DemandRigid
- DemandInelastic
- DemandElastic
- DemandUElastic – unitary elastic **demand**

Offers:

- OfferInelastic
- OfferElastic

- OfferUElastic
- OfferPIelastic

Goods (as a function of income):

- GoodInferior
- GoodNecessary
- GoodSuperior

Advantages of this approach:

- **The code is modular:** each class handles a single example type.
- The application can be **easily extended**: a new class (BunDeLux, etc.) can be added without modifying the interface.
- **The Open/Closed principle is respected**: existing code does not need to be modified to introduce a new behavior.

5. Testing

5.1 Purpose of testing

The testing aimed to verify the correctness of the essential functionalities implemented in the external libraries (DLLs), which contain the calculation logic of the MiniContaBill application. These modules include economic calculations such as capital depreciation, profit estimation, elasticities determination and salary calculation. Since these components are independent of the graphical interface, they could be tested in isolation, ensuring the accuracy of the generated results.

5.2 Methodology

The tests were performed manually, by directly calling the methods in the DLLs with valid data sets. For each main function, the returned results were checked for different combinations of inputs, to confirm that the implementation of the formulas is correct.

No tests for invalid or extreme values were included, because the input validation is done at the graphical interface level (Forms), and the logic in the DLL assumes that the received data is already verified.

5.3 Test Cases

5.3.1 FirmUtil DLL

Nr. Test	Valoare	Raspuns corect	Raspuns program	Observatii
1 returnare tip firma	SRL	FirmType.SRL	FirmType.SRL	corect
2 returnare tip firma	PFA	FirmType.PFA	not FirmType.SRL	corect
3 setare CA firmă	1000	1000	1000	corect
4 setare nume firmă	Bilibic	Bilibic	Bilibic	corect
5 setare actiuni la o firmă ce nu poate avea actiuni	100	0	0	corect

5.3.2 SalariiAngajati DLL

Nr Test	Valoare	Rezultat corect	Rezultat program	Observ atii
1.CalculateAverageSalary	5000, 4500, 7000	5500.00	5500.00	Corect
2.CalculateTotalSalary	5000, 4500, 7000	16500.00	16500.00	Corect
3.CalculateNetSalary	SalariuBrut = 5000	3275.00	3275.00	Corect
4.CalculateTotalEmployerCost	SalariuBrut = 5000	5112.50	5112.50	Corect
5.CalculateAverageSalary	[]	0	0	Corect
6.CalculateNetSalary	null	0	0	Corect

5.3.3 CostProductAndProfit DLL

Nr. Test	Valoare	Raspuns corect	Raspuns program	Observatii
1 returnare masa profit	("1000", "600")	"400.00"	"400.00"	corect
2 returnare rata profit	("200", "800")	"25.00"	"25.00"	corect
3 returnare pret final	("80", "20")	"100.00"	altul \neq "100.00"	testul verifică diferența, deci funcția e corectă
4 verificare prag rentabilitate pozitiv	("100", "500", "", "50", true)	true	true	corect
5 verificare impozit profit pentru PFA	("1000", true)	"100.00"	"100.00"	corect

5.3.4 DemandAndSupply DLL

Nr. Test	Valoare	Răspuns corect	Răspuns program	Observații
1 returnare obiect Cerere Elastică	"Cerere elastică"	cerereElastica.generateExample() .typeOfElasticity	cerereElastica.generateExample().typeOfElasticity	corect
2 returnare obiect Cerere Inelastică	"Cerere inelastică"	cerereInelastica.generateExample() .typeOfElasticity	cerereInelastica.generateExample().typeOfElasticity	corect
3 returnare obiect BunInferior	"Bun inferior"	bunInferior.generateExample() .typeOfElasticity	bunInferior.generateExample().typeOfElasticity	corect
4 returnare obiect BunNecesar	"Bun necesar"	bunNecesar.generateExample().typeOfElasticity	bunNecesar.generateExample().typeOfElasticity	corect
5 returnare obiect BunSuperior	"Bun superior"	bunSuperior.generateExample().typeOfElasticity	bunSuperior.generateExample().typeOfElasticity	corect

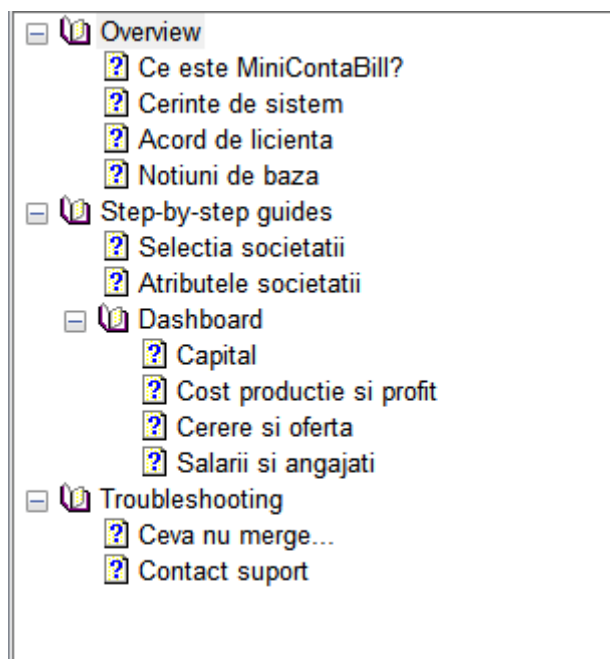
5.4 Conclusion

All functions in the DLLs were tested with correct data and returned results as expected. The testing focused on verifying the implemented formulas and correct behavior under normal usage conditions. Given the clear separation between interface and logic, testing at the DLL level was considered sufficient to validate the application's functionalities.

7. User Guide

7.1 Help file organization (CHM)

MiniContaBill includes a complete help system, in CHM format, which provides step-by-step instructions for all the application's functionalities. The guide structure is logically organized (Overview, Step-by-step Guides, Dashboard, Troubleshooting), and users can quickly access relevant sections through the application interface.



7.2 Overview


The “Overview” page in the MiniContaBillHelp.chm help file provides an overview of the application and how to use it. It is the first screen displayed when the user accesses the in-app guide and is the recommended starting point for anyone using MiniContaBill for the first time.

The introductory text highlights the fact that the application is easy to use, being dedicated especially to small entrepreneurs. The help document is designed to support both new users and those who want to deepen their knowledge.

The recommended sections for getting started, as they appear on this page, are:

- What is MiniContaBill? – description of the application’s purpose and functionalities
- System requirements – system compatibility check
- License agreement – legal information regarding use
- Basics – guide to navigating the application
- Step-by-step guides – detailed instructions for common actions
- Troubleshooting – solutions to common problems

This page helps the user to easily navigate the content of the guide and quickly identify relevant sections.

Overview


Bine ai venit în MiniContaBill!

Aceasta este o aplicație ușor de utilizat și intuitivă, destinată să vină în ajutorul micilor întreprinzători.

Acest document de tip *help* este conceput pentru a te ajuta să înveți rapid cum să folosești MiniContaBill – fie că ești un utilizator nou, fie că dorești să-ți aprofundezi cunoștințele ca utilizator obișnuit.

Pentru început, îți recomandăm să consulți următoarele secțiuni:

- [Ce este MiniContaBill?](#)
Află mai multe despre scopul și funcționalitățile aplicației.
- [Cerințe de sistem](#)
Verifică dacă dispozitivul tău este compatibil cu MiniContaBill.
- [Acord de licență](#)
Informații legale despre utilizarea aplicației.
- [Noțiuni de bază](#)
Ghid introductiv pentru navigarea în aplicație și folosirea principalelor funcții.
- [Step-by-step guides](#)
Instrucțiuni detaliate pentru cele mai comune acțiuni și fluxuri de lucru.
- [Troubleshooting](#)
Soluții pentru probleme frecvente și întrebări comune.

7.3 Step-by-step Guides

The “Step-by-step guides” section in the MiniContaBillHelp.chm file provides users with a set of detailed instructions for the most important actions in the application. Each guide explains, in a clear and accessible way, the steps necessary to correctly use the functionalities offered by MiniContaBill.

The guides are designed to be easy to follow, regardless of the user's experience level. They cover the basic steps, from selecting the company, to using each main module in the application.

The available step-by-step guides include:

- Company selection – choosing the active company and uploading the JSON file
- Company attributes – filling in company-specific information
- Dashboard – using the functional modules:
 - Capital
 - Production cost and profit
 - Demand and supply
 - Salaries and employees

These guides can be accessed directly from the application via the Help button, and provide support in carrying out each main action.

Step-by-step guides ◀ ▶

În această secțiune poți urmări, pas cu pas, cum se utilizează aplicația MiniContaBill. Ghidurile sunt create astfel încât orice utilizator, indiferent de nivelul de experiență, să poată folosi aplicația.

Le poți accesa astfel:

- [Selecția societății](#)
- [Atributele societății](#)
- [Dashboard](#)
 - [Capital](#)
 - [Cost producție și profit](#)
 - [Cerere și ofertă](#)
 - [Salarii și angajați](#)

7.3.1 Company selection

The first step in using the MiniContaBill application is to enter the company's identification data. The user must select the company type (SRL, SA or PFA) and manually fill in the company name.



La deschiderea aplicației, după apăsarea butonului de start vei întâmpina ecranul de selecție a societății.

1. **Textbox:** Alegere denumire firmă
2. **Dropdown list:** Alegere tip societate comercială(SRL, SA, PFA)

3. **Buton next:** Permite accesul la următorul ecran.

7.3.2 Company Attributes

After selecting the company type (SRL, SA or PFA) and entering the company name, the user must complete a series of company-specific attributes. This information includes: share capital, turnover, tax code, debts and, in the case of SA companies, the number of shares and their value.

Atributele societatii

[Step-by-step guides >>](#)



În acest ecran se selectează valorile atributelor firmei.

1. Atribute firmei unde inserăm attributele
2. **Buton back:** Permite revenirea la [ecranul anterior](#)
3. **Buton next:** Permite accesul la [următorul ecran](#)

7.3.3 Dashboard

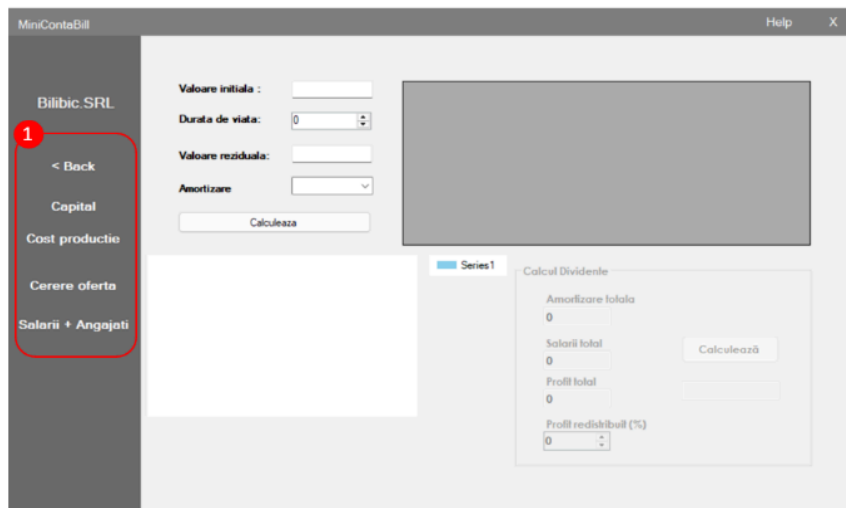
The Dashboard represents the centralized interface of the MiniContaBill application, providing the user with a quick and intuitive way to access the main functionalities. It is designed to simplify navigation between the different components of the application, allowing for efficient and organized use.

Dashboard

Step-by-step guides >>



Dashboard-ul este o interfață centralizată care oferă utilizatorului o modalitate rapidă și intuitivă de acces la principalele funcționalități ale aplicației. Este conceput pentru a simplifica navigarea între diversele module ale sistemului.



1. Panoul de control se regăsește în partea din stânga a dashboard-ului. El conține o serie de butoane utile pentru navigare și acces direct către diferite funcționalități. De sus în jos, acestea sunt:

- **Buton back:** Permite revenirea la [ecranul anterior](#) din aplicație
- **Aplicații Capital:** Deschide secțiunea destinată [gestionării aplicațiilor de capital](#)
- **Aplicații Cost Producție:** Oferă acces la modulele de [analiză a costurilor de producție și profitul](#)
- **Aplicații Cerere și Ofertă:** Deschide tab-ul în care se pot [analiza cererile și ofertele](#)
- **Aplicații Salarii și Angajați:** Direcționează către funcționalitățile legate de [managementul resurselor umane și salarizare](#)

• Capital

The Capital module is dedicated to managing fixed assets and calculating their depreciation. The user enters information such as the initial value of the asset, its useful life and residual value, then selects the desired depreciation method:

- Linear
- Decreasing
- Progressive

After filling in the data and pressing the Calculate button, the application displays:

- a table with the depreciation values for each year,
- a graph representing the evolution of the asset value.

Dividend Calculation

For SA type companies, the application also offers the possibility to calculate the dividends to be distributed to shareholders. The user enters:

- Total profit obtained

- Percentage of profit allocated to dividends
- Number of shares

The application calculates:

- Total dividends distributed
- Dividend per share

This functionality is available only if the selected company is SA type, and the fields related to shares are completed.

Capital
Step-by-step guides » Dashboard »

Această secțiune permite gestionarea capitalului disponibil în cadrul organizației. Utilizatorii pot vizualiza, analiza amortizarea cât și calcula dividendul.

1

Valoare inițială : 1000

Durata de viață : 6

Valoare reziduală : 200

Amortizare : Progresiva

Calculează

2

An	Amortizare Anuală	Valoare Ramasă
1	38.1	961.9
2	76.19	885.71
3	114.29	771.43
4	152.38	619.05
5	190.48	428.57
6	228.57	200

3

Amortizare anuală

Amortizare progresiva

4

Calcul Dividende

Amortizare totală : 0

Salarii total : 0

Profit total : 0

Profit redistribuit (%) : 0

Calculează

Funcționalități principale:

1. **Alocarea și calcularea amortizării:** Calculează amortizarea. Ea poate fi de trei tipuri (liniară, progresivă, degresivă). Ea încarcă atât tabelul cât și graficul odată ce butonul calculează este apăsat



2. **Vizualizare amortizare tabel:** Posibilitatea de a vizualiza cum se modifică valoarea amortizării în timp
3. **Vizualizare amortizare grafic:** Posibilitatea de a vizualiza cum se modifică valoarea amortizării față de alți ani
4. **Calcul dividende:** Generează dividendului obținut dintr-o acțiune (pentru calcul se folosesc și valori din ecranul anterior)

● Production Cost and Profit

This module allows estimating the production costs and profit of a product or service. The user fills in the following information:

- Production cost
- Desired profit margin
- Applicable VAT
- Sales volume

The application automatically calculates:

- Sales price with and without VAT,
- Profit obtained,
- Break-even point (equilibrium).

The module provides a clear picture of the product's profitability.

Cost productie si profit
Step-by-step guides » Dashboard »

Această secțiune este dedicată gestionării costurilor asociate proceselor de producție. Oferă o imagine asupra cheltuielilor optimizării resurselor. Pentru a putea vizualiza întreaga funcționalitate a tab-ului folosiți scrollbar-ul (vedeți dreapta) sau roțița de la mouse.

1. Masă profit

Preț vânzare / Venituri: Masă profit:
Costuri: Calculează

2. Rată profit

Preț vânzare / Venituri: Rată profit:
Costuri / CA: Calculează

3. Preț final

Chiria: Materiale Prime: Materiale Auxiliare: Cost Total:
Amortizare: Salarii: Calculează Cost Mediu:
Profit Mediu (%CM): Nr unități produse: Preț Final:
Pie chart showing: (50.0%), (10.0%), (10.0%), (10.0%), (10.0%)

4. Prag de rentabilitate

a) atinge prag de rentabilitate b) pentru profit (500k unități)

Preț unitate produs: Cost Variabile Medii: Profit:
Cost fix total: Calculează Cantitate:
Bar chart showing: 0, 1, 2, 3, 4, 5

5. Impozit pe profit

a) cu PFA b) fără PFA

Profit: Calculează Impozit pe profit:

6. Preț cu TVA

Preț brut: Cota TVA: Preț cu TVA:
Calculează

Funcționalități principale:

1. **Calcul masă profit:** Determinarea masă profit pentru fiecare produs
2. **Calcul rată profit:** Determinarea rată profit pentru fiecare produs
3. **Calcul preț final:** Urmărirea costurilor de producție a produsului printr-un pie chart cât și calcularea prețului final, cost mediu și total
4. **Calcul prag de rentabilitate:** Compararea costurilor pentru a urmări câte produse trebuie vândute pentru a atinge pragul de rentabilitate sau pentru a ajunge la profit (la alegere prin selecția a) sau b))
5. **Calcul impozit pe profit:** Determinarea impozitului pe profit (în funcție de tipul de societate o opțiune este deja aleasă și nu se poate schimba)
6. **Calcul preț cu TVA:** Determinarea prețului după aplicarea TVA-ului

● Demand and Supply

In this module, the user can analyze the behavior of the market depending on the variation of price, income or other factors. It is possible to calculate:

- Elasticity of demand with respect to price
- Elasticity of demand with respect to income
- Elasticity of supply

Initial and final values for price (P0, P1), quantity (Q0, Q1) or income (V0, V1) are entered, and the application displays:

- the value of elasticity,
- its economic interpretation (inelastic, elastic, etc.),

Cerere si oferta

Step-by-step guides » Dashboard »



Secțiunea Cerere și Ofertă permite analizarea echilibrului între ceea ce este solicitat de piață și ce oferă compania. Este un instrument important pentru decizii strategice și de vânzări.

Funcționalități principale:

1. **Elasticitatea cererii:** Afișează cât de rentabilă este modificarea prețului sau a venitului

La selecția acestui câmp se pot vedea următoarele după ce s-a ales (în funcție de preț sau venit):

La apăsarea **butonului Calculează** se va afișa print-un pop-up tipul elasticității și al bunului.

Pentru o înțelegere mai bună a funcționalității puteți apăsa pe **butonul Exemplu** unde se vor putea vedea următoarele:

Unde se poate selecta unul din cele 4 tipuri de exemple.

2. Elasticitatea ofertei: Afișează cât de rentabilă este modificarea ofertei

La selecția acestui câmp se pot vedea următoarele

The screenshot shows a web form titled "Elasticitatea ofertei". It contains four input fields arranged in a 2x2 grid:

cantitate oferită inițial	cantitate oferită după modificare	preț inițial	preț după modificare
100	140	10	11

Below the input fields are two buttons: "Exemplu" and "Calculează".

La apăsarea **butonului Calculează** se va afișa print-un pop-up tipul elasticității.

Pentru o înțelegere mai bună a funcționalității puteți apăsa pe **butonul Exemplu** unde se vor putea vedea următoarele:

The screenshot shows the same web form titled "Elasticitatea ofertei", but instead of input fields, it displays four buttons representing different types of elasticity:

- Ofertă elastică
- Ofertă inelastică
- Ofertă perfect inelastică
- Ofertă unitar elastică

Unde se poate selecta unul din cele 4 tipuri de exemple.

• Salaries and Employees

The Salary and Employees module is responsible for managing the company's personnel. The user can:

- Add, edit or delete employees in a dedicated table
- Calculate the net salary for each employee, based on the entered gross salary
- Display:
 - Average salaries
 - Total cost to the employer
 - Total number of employees
- Save or load employee data in a JSON file

This functionality allows for efficient tracking of personnel costs and keeping data organized, per company.

Salarii si angajati

Step-by-step guides » Dashboard »



Această aplicație permite gestionarea resurselor umane, a salariilor, și a evidenței personalului. Este esențială pentru salarizare.

The screenshot shows the application interface with four numbered callouts:

- 1** Points to a table displaying employee data:

Id	NumeAngajat	Funcție	SalariuBrut	Departament
1	Ion	președinte	5000	primar
2	Vasilica	spălatoarea	700	curatorie

- 2** Points to input fields for adding a new employee:

Id	Nume Angajat	Funcție	Salariu Brut	Departament
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

- 3** Points to summary statistics:

Salariu Mediu	Total Salarii	Salariu Net	Cost total angajator
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

- 4** Points to action buttons:

<input type="button" value="Adauga angajat"/>	<input type="button" value="Sterge angajat"/>	<input type="button" value="Actualizeaza angajat"/>	<input type="button" value="Incarca Lista Angajati"/>	<input type="button" value="Salveaza Lista Angajati"/>
<input type="button" value="Salariu mediu"/>	<input type="button" value="Total salarii"/>	<input type="button" value="Calcul Salariu Net"/>	<input type="button" value="Cost Total Angajator"/>	

Funcționalități principale:

- 1. Tabel vizualizare angajați:** Vizualizarea sub formă de tabel a datelor angajaților
- 2. Câmp proprietăți salariat:** Contruiește unui angajat cu proprietățile din câmp
- 3. Câmp vizualizare date despre totalitatea angajaților:** Vizualizarea unor atribute calculate din totalitatea angajaților
- 4. Câmp gestionare salariat:** Realizează o funcționalitate descrisă de denumirea butonului ales
 - **Buton Adaugă angajat:** Adaugă un angajat în tabel
 - **Buton Sterge angajat:** Sterge un angajat în tabel
 - **Actualizează angajat:** Actualizează un angajat din tabel
 - **Buton Incarcă Listă Angajați:** Incarcă lista de angajați dacă există
 - **Buton Salvează Listă Angajați:** Salvează lista curentă de angajați
 - **Buton Salariu mediu:** Calculează salariul mediu
 - **Buton Total salarii:** Calculează valoarea totală a salariilor

7.4 Troubleshooting

The Troubleshooting section in the MiniContaBillHelp.chm help file provides solutions to the most common problems that may arise while using the application. The purpose of this section is to support the user in quickly identifying and resolving errors, without advanced technical intervention.

For problems that cannot be resolved with the suggestions above, the user can contact the application administrator.

Ceva nu merge...

Troubleshooting »



1. Anumite câmpuri sunt disabled

Exemplu:

Atribute societate

Cifra de afaceri:

Capital fix:

Capital circulant:

Numar actiuni:

Valoare actiuni:

Datorii:

Calcul Dividende

Amortizare totala

Salarii total

Profit total

Profit redistribuit (%)

Calculează

sau

Cauze:

- Tipul firmei nu corespunde cu funcționalitatea ce are nevoie de astfel de câmpuri (Comportament implementat în aplicație, nu este eroare)

Soluții:

- Schimbă [tipul firmei](#)

2. Aplicația nu pornește

Exemplu: Dublu-clic pe aplicație, dar fereastra nu se deschide sau se închide imediat.

Cauze posibile:

- Lipsă fișiere necesare în folderul aplicației
- .NET Framework neinstalat sau corupt
- Permisuni insuficiente

Soluții:

- Verifică dacă toate fișierele aplicației sunt prezente
- Asigură-te că este instalat .NET Framework versiunea 4.8
- Rulează aplicația ca administrator
- Reinstalează aplicația

Exemplu: Panoul de control sau tab-urile nu sunt afișate complet

Cauze posibile:

- Rezoluția ecranului este prea mică
- Există o eroare internă de afișare

Soluții:

- Asigură-te că folosești o rezoluție minimă recomandată ([vezi aici](#))
- Repornește aplicația

4. Datele nu se salvează

Exemplu: După completarea tabelului și apăsare butonului **Salvează Listă Angajați**, datele nu sunt salvate

Cauze posibile:

- Nu s-a apăsă butonul **Salvează Listă Angajați**
- Eroare de validare a câmpurilor

Soluții:

- Verifică dacă ai completat toate câmpurile
- Reîncearcă a salvare precedentă (dacă există)
- Repornește aplicația

5. Butoanele din dashboard nu răspund

Exemplu: La apăsarea pe butoane (ex. din "Capital"), nu se întâmplă nimic

Cauze posibile:

- Aplicația este blocată sau are un delay
- Eroare în codul de legătură

Soluții:

- Închide și redeschide aplicația

Dacă problemele persistă contactează [administratorul aplicației](#).

Contact suport

Troubleshooting »

Dacă problemele tale persistă, te rugăm să contactezi echipa de suport tehnic:

- **Email:** suport@firma.ro
- **Telefon:** 0123 456 789
- **Program:** Luni – Vineri, 09:00 – 17:00

8. Conclusions

The MiniContaBill project aimed to develop an intuitive desktop application, intended for small entrepreneurs, for performing basic economic calculations. The application integrates modules for capital analysis, cost and profit estimation, economic elasticities calculation and employee salary management.

In the development process, object-oriented design principles were applied, using design patterns such as Builder (for building Company objects) and Strategy (for generating elasticity examples). The application logic is separated from the interface, being organized in reusable libraries (DLLs).

Testing was performed on functions from external libraries, with results in line with theoretical expectations. The application also includes an integrated help system (CHM format), which provides step-by-step guides and solutions for potential problems.

The project demonstrates a clear, modular and easily extensible architecture, which meets the functional requirements initially established. The development experience provided a practical deepening of theoretical concepts in the field of programming applied to economics.

9. Possible future developments

To improve and extend the functionality of the MiniContaBill application, the following future development directions can be considered:

- Data persistence in a relational database (e.g. SQL Server) instead of JSON files, for better scalability and centralized management of companies and employees.
- Automatic export of financial reports in PDF or Excel format, useful for accounting and presentation.
- Account-based authentication, for multiple user management and data protection.
- Web or mobile interface, allowing access to the application from different devices.

- Integration with fiscal APIs, for automatic obtaining of data on contributions and VAT rates.
- Automatic unit testing, through frameworks such as NUnit or xUnit, to increase the robustness of the application in case of future changes.

These extensions would transform MiniContaBill into an even more powerful tool for economic analysis and financial management of small and medium-sized companies.

Appendix 1 - Relevant code snippets

A1. Depreciation calculation (3 methods: linear, progressive, declining balance)

File: TabCapitalForm.cs

```
private void buttonCalcAmortizare_Click(object sender, EventArgs e)
{
    double valoareInitiala;
    int durataViata = Convert.ToInt32(Math.Round(numericUpDownLifeTime.Value,
0));
    double valoareaReziduala;

    // Validare valori
    if (!double.TryParse(textBoxInitialValue.Text, out valoareInitiala) ||
        !double.TryParse(textBoxRezidualValue.Text, out valoareaReziduala))
    {
        MessageBox.Show("Valori invalide.");
        return;
    }

    if (comboBoxAmortizare.SelectedIndex == -1)
    {
        MessageBox.Show("Selectați o metodă de amortizare.");
        return;
    }

    List<Amortizare> amortizari;

    // Alegerea metodei de calcul în funcție de selecție
    switch (comboBoxAmortizare.SelectedIndex)
    {
        case 0:
            amortizari = AmortizareCalculator.Liniara(valoareInitiala,
durataViata, valoareaReziduala);
            break;
        case 1:
            amortizari = AmortizareCalculator.Progresiva(valoareInitiala,
durataViata, valoareaReziduala);
```

```

        break;
    case 2:
        amortizari = AmortizareCalculator.Degresiva(valoareInitiala,
durataViata, valoareaReziduala);
        break;
    default:
        return;
    }

    dataGridViewAmortizari.DataSource = amortizari;
}

```

A2. Dividend calculation (only for SA)

File:TabCapitalForm.cs

```

    private void buttonCalc_Click(object sender, EventArgs e)
    {
        try
        {
            decimal amortizareTotala =
ConvertText.ConvertText.ConvertToDecimal(textBoxAmortizareTotala.Text);
            decimal salariiTotal =
ConvertText.ConvertText.ConvertToDecimal(textBoxSalariiTotal.Text);
            decimal ct =
Dividente.Dividente.CalculCheltuieliTotale(amortizareTotala,
FormManager.builder.GetResult().GetCC(), salariiTotal);
            decimal profit =
Dividente.Dividente.CalculProfit(FormManager.builder.GetResult().GetCA(), ct);
            decimal rezultat = Dividente.Dividente.CalculDivident(profit,
numericProcent.Value, FormManager.builder.GetResult().GetNrAct());

            if (rezultat <= 0)
            {
                MessageBox.Show("Rezultatul calculului este zero sau negativ.
Verificați datele introduse.", "Atenție!", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return;
            }

            textBoxDivident.Text = rezultat.ToString("F2");
        }
        catch (FormatException)
        {
            MessageBox.Show("Introduceți valori numerice valide în toate
câmpurile.", "Eroare format", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        catch (Exception ex)
        {
            MessageBox.Show($"A apărut o eroare: {ex.Message}", "Eroare",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

```



```
}
```

A3. Calculating price elasticity of demand

File: TabCerereSiOfertaForm.cs

```
private void hopeButton1_Click(object sender, EventArgs e)
{
    try
    {
        float qc0, qc1, p0, p1;
        string c = "";
        if (float.TryParse(hopeTextBox1.Text, out qc0) == false ||
            float.TryParse(hopeTextBox2.Text, out qc1) == false ||
            float.TryParse(hopeTextBox3.Text, out p0) == false ||
            float.TryParse(hopeTextBox4.Text, out p1) == false)
        {
            throw new FormatException();
        }
        float ecp = ((qc1 - qc0) / qc0) / ((p1 - p0) / p0);
        switch (hopeComboBox1.SelectedItem.ToString())
        {
            case "în funcție de preț":
                if (ecp == 0)
                {
                    c = "\nCerere rigidă.";
                }
                else if (ecp == -1)
                {
                    c = "\nCerere unitar elastică.";
                }
                else if (ecp < -1)
                {
                    c = "\nCerere elastică";
                }
                else if (ecp > -1 && ecp < 0)
                {
                    c = "\nCerere inelastică sau slab elastică.";
                }
                break;
            case "în funcție de venit":
                if (ecp < 0)
                {
                    c = "\nBun inferior.";
                }
                else if (ecp > 0 && ecp < 1)
                {
                    c = "\nBun necesar.";
                }
                else if (ecp > 1)
```

```

        {
            c = "\nBun superior.";
        }
        break;
    }

    System.Windows.Forms.MessageBox.Show("Elasticitatea cererii este: " +
    ecp.ToString() + c, "Rezultat");
}
catch (FormatException)
{
    System.Windows.Forms.MessageBox.Show("Valori incorecte!", "EROARE!");
}
}

```

A4. Calculating Net Salary

File: TabSalariiForm.cs

```

private void buttonSalariuNet_Click(object sender, EventArgs e)
{
    if (_angajati.Count == 0)
    {
        MessageBox.Show("Nu ati selectat un angajat pentru a calcula
salariul NET.");
    }

    if (_index >= 0 && _index < _angajati.Count)
    {
        var angajat = _angajati[_index];

        if (angajat.SalariuBrut < 6050)
        {
            MessageBox.Show("Angajatul are un salariu brut mai mare decat
salariul minim brut plus 2000 lei, deci are deducere personală.");
        }

        double
netSalary=SalaryCalculator.CalculateNetSalary(_angajati[_index]);
        textBoxNetSalary.Text = netSalary.ToString("F2");
    }
}

```

A5. Break-even point calculation + graphic display

File: TabCostProfitForm.cs

```
private void buttonCalculeazaPregdeRentabilitate_Click(object sender,
EventArgs e)
{
    try
    {
        var calculator = new PragRentabilitate();
        string rezultat =
calculator.CalculeazaPragRentabilitate(textBoxPretUnitateProdus.Text,
textBoxCostFix.Text, textBoxProfit1.Text, textBoxCostVariabileMedii.Text,
radioButtonRentabilitateCazA.Checked);
        textBoxCantitate.Text = rezultat;
        SetupBarChartPragRenta(radioButtonRentabilitateCazA.Checked);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Eroare la calcul: {ex.Message}", "Eroare",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        textBoxCantitate.Text = "0";
    }
}
```

A6. Function for Calculating Linear Depreciation

File: Amortizari.cs (DLL)

```
public static List<Amortizare> Liniara(double initialValue, int lifeTime,
double rezidualValue)
{
    if (initialValue <= 0)
        throw new ArgumentException("Valoarea inițială trebuie să fie un număr
pozitiv.");

    if (lifeTime <= 0)
        throw new ArgumentException("Durata de viață trebuie să fie un număr
întreg pozitiv.");

    if (rezidualValue < 0)
        throw new ArgumentException("Valoarea reziduală nu poate fi
negativă.");

    if (rezidualValue >= initialValue)
        throw new ArgumentException("Valoarea reziduală trebuie să fie mai
mică decât valoarea inițială.");

    var amortizari = new List<Amortizare>();
    double amortizareAnuala = (initialValue - rezidualValue) / lifeTime;
```

```

double valoareRamasa = initialValue;

for (int an = 1; an <= lifeTime; an++)
{
    valoareRamasa -= amortizareAnuala;
    amortizari.Add(new Amortizare
    {
        An = an,
        AmortizareAnuala = amortizareAnuala,
        ValoareRamasa = valoareRamasa
    });
}
return amortizari;
}

```

A7. Function for Calculating Net Salary

File: Angajati.cs (DLL)

```

public static double CalculateNetSalary(Angajat angajat)
{
    if (angajat == null)
        return 0;

    // CAS - contribuția la asigurări sociale (25%)
    double asigurareSociala = 0.25 * angajat.SalariuBrut;

    // CASS - contribuția la asigurări de sănătate (10%)
    double asigurareSanatate = 0.10 * angajat.SalariuBrut;

    // Impozit pe venit (10% din venitul rămas după CAS și CASS)
    double impozit = 0.10 * (angajat.SalariuBrut - asigurareSociala -
    asigurareSanatate);

    //Salariu Net rezultat
    return angajat.SalariuBrut - asigurareSociala - asigurareSanatate -
    impozit;
}

```

A8. Class for Calculating Price with VAT

File: CostulProductieProfitDLL.cs

```

public class PretTva
{
    /// <summary>
    /// Calculeaza pretul final al unui produs inclusiv TVA.
    /// </summary>

```

```

    /// <param name="pretBrut">Pretul initial al produsului, fara TVA</param>
    /// <param name="cotaTva">Procentul de TVA exprimat in format
zecimal</param>
    public decimal CalculeazaPretCuTva(decimal pretBrut, decimal cotaTva)
    {
        return pretBrut + cotaTva * pretBrut;
    }

    /// <summary>
    /// Supraincarcare a metodei de calcul pentru TVA care accepta valori sub
forma de text.
    /// </summary>
    /// <param name="pretBrutText">Textul reprezentand pretul brut</param>
    /// <param name="cotaTvaText">Textul reprezentand cota TVA</param>
    /// <returns>Pretul final cu TVA formatat cu doua zecimale</returns>
    /// <exception cref="ArgumentException">Exceptie aruncata cand valorile
introduse nu pot fi convertite in numere</exception>

    public string CalculeazaPretCuTva(string pretBrutText, string
cotaTvaText)
    {
        if (decimal.TryParse(pretBrutText, out decimal pretBrut) &&
            decimal.TryParse(cotaTvaText, out decimal cotaTva))
        {
            decimal rezultat = CalculeazaPretCuTva(pretBrut, cotaTva);
            return rezultat.ToString("F2");
        }
        throw new ArgumentException("Valorile introduse trebuie să fie numere
valide");
    }
}

```

A9. Implementing Builder for SRL type companies

```

public class SRLBuilder : IFirmBuilder
{
    public Firm firm = new Firm();

    /// <inheritdoc/>
    public void SetName(string name) => firm.name = name;

    /// <inheritdoc/>
    public void SetCA(uint ca) => firm.ca = ca;

    /// <inheritdoc/>
    public void SetCF(uint cf) => firm.cf = cf;

    /// <inheritdoc/>
    public void SetCC(uint cc) => firm.cc = cc;

    /// <summary>
    /// Metodă ce setează cu 0 numărul de acțiuni

```

```

    /// </summary>
    /// <param name="nrActiuni">Numărul de acțiuni</param>
    public void SetNrActiuni(uint nrActiuni) => firm.nrActiuni = 0;

    /// <summary>
    /// Metodă ce setează cu 0 prețul unei acțiuni
    /// </summary>
    /// <param name="nrActiuni">Valoarea unei acțiuni</param>
    public void SetPretActiune(uint pretActiune) => firm.pretActiune = 0;

    /// <inheritdoc/>
    public void SetDatorii(uint datorii) => firm.datorii = datorii;

    /// <inheritdoc/>
    public Firm GetResult()
    {
        firm.type = FirmType.SRL;
        return firm;
    }
}

```

A10. Implementing Strategy Pattern for "Good Inferior"

```

public class BunInferior : ICerereStrategy
{
    public CerereValori generateExample()
    {
        CerereValori e = new CerereValori();
        e.typeOfElasticity = "Bun inferior";
        e.explanation = "Când venitul crește, oamenii renunță la acest bun
și preferă alternative mai bune.\n" +
            "După formula  $E_{cp} = ((Q_{c1} - Q_{c0}) / Q_{c0}) / ((V_1 - V_0) / V_0)$  rezultă
 $E_{cp} = (-0.2) / 0.2 = -1.$ ";
        e.qc0 = "100";
        e.qc1 = "80";
        e.p0 = "1000";
        e.p1 = "1200";
        return e;
    }
}

```